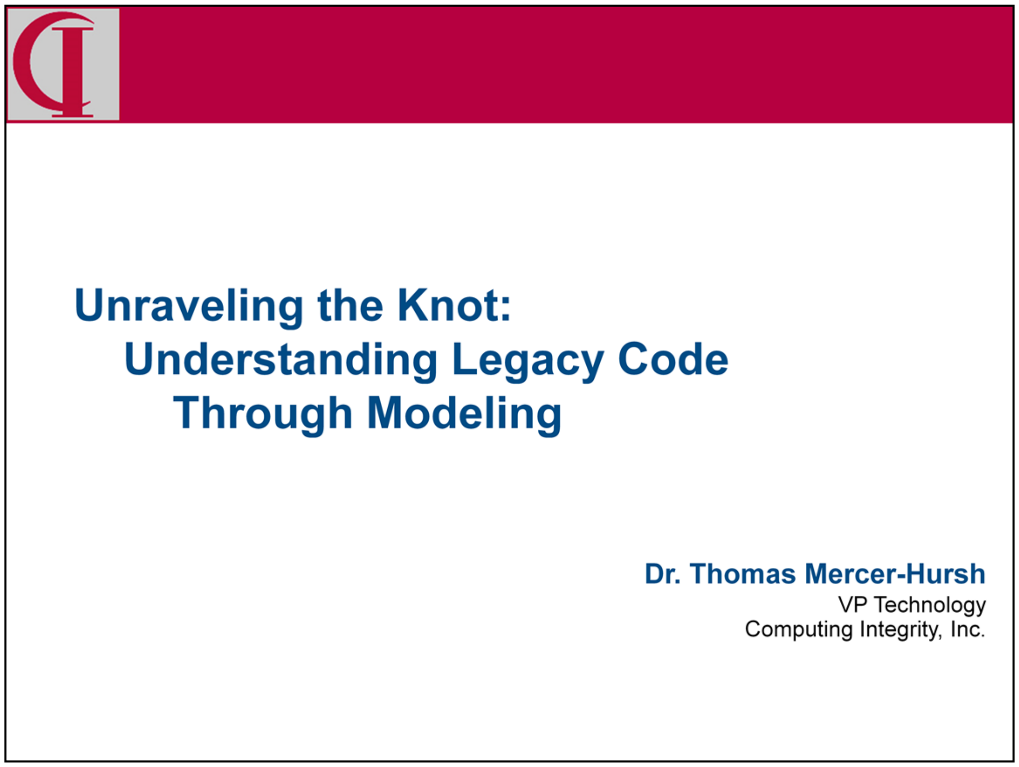
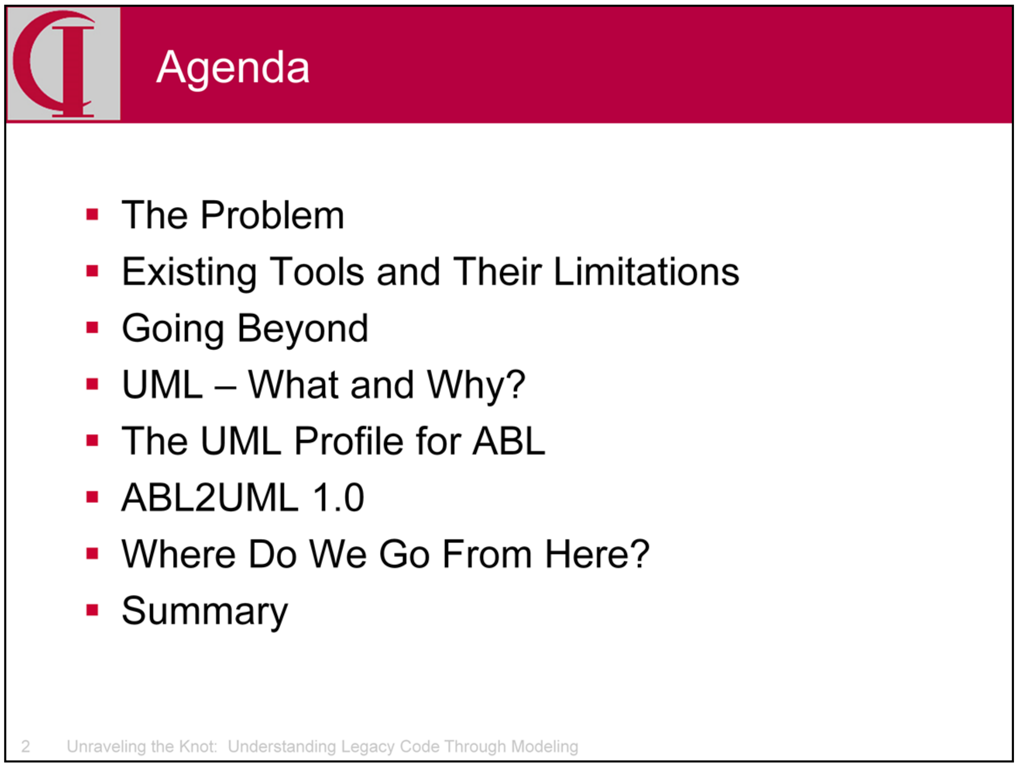


Unraveling the Knot: Understanding Legacy Code Through Modeling

Dr. Thomas Mercer-Hursh
VP Technology
Computing Integrity, Inc.

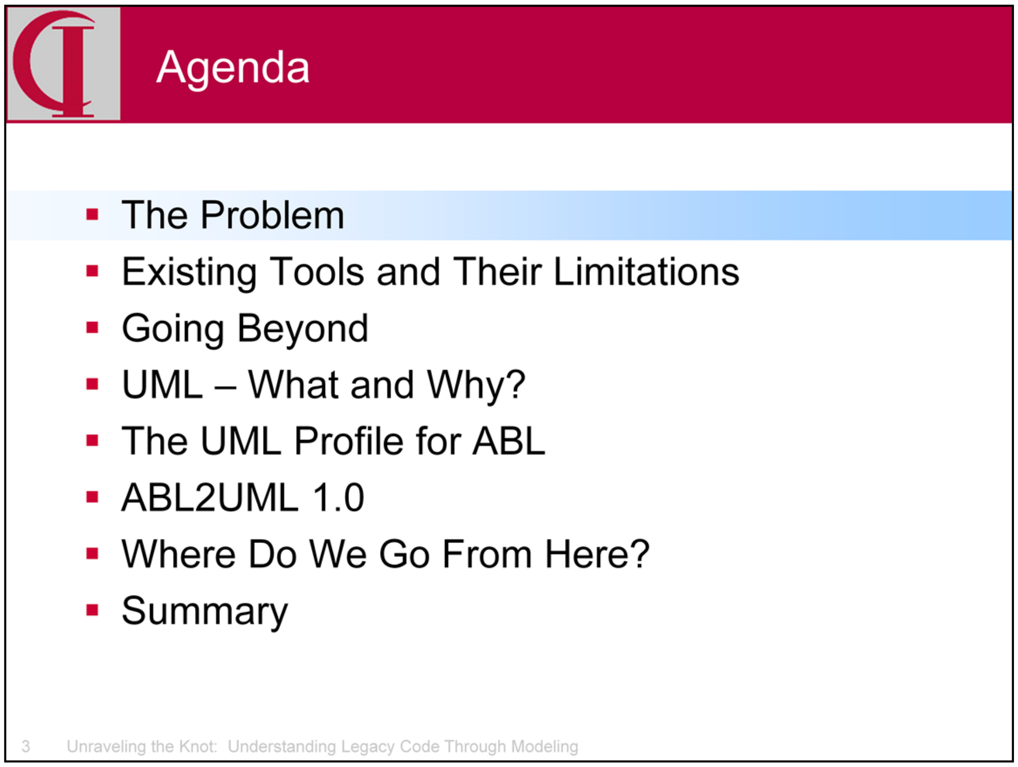
Let me begin by introducing myself. I have been a Progress Application Partner since 1986 and for many years I was the architect and chief developer for our ERP application. In recent years I have refocused on the problems of transforming and modernizing legacy ABL applications. This has led me to exploring UML as a tool in the transformation process. Today's talk is to tell you about my efforts to analyze existing applications using UML.



Agenda

- The Problem
- Existing Tools and Their Limitations
- Going Beyond
- UML – What and Why?
- The UML Profile for ABL
- ABL2UML 1.0
- Where Do We Go From Here?
- Summary

Here is our agenda for today. First we are going to talk a little bit about the background, then about what UML offers, a UML Profile for ABL, the initial version of ABL2UML, and then look forward to work in process.

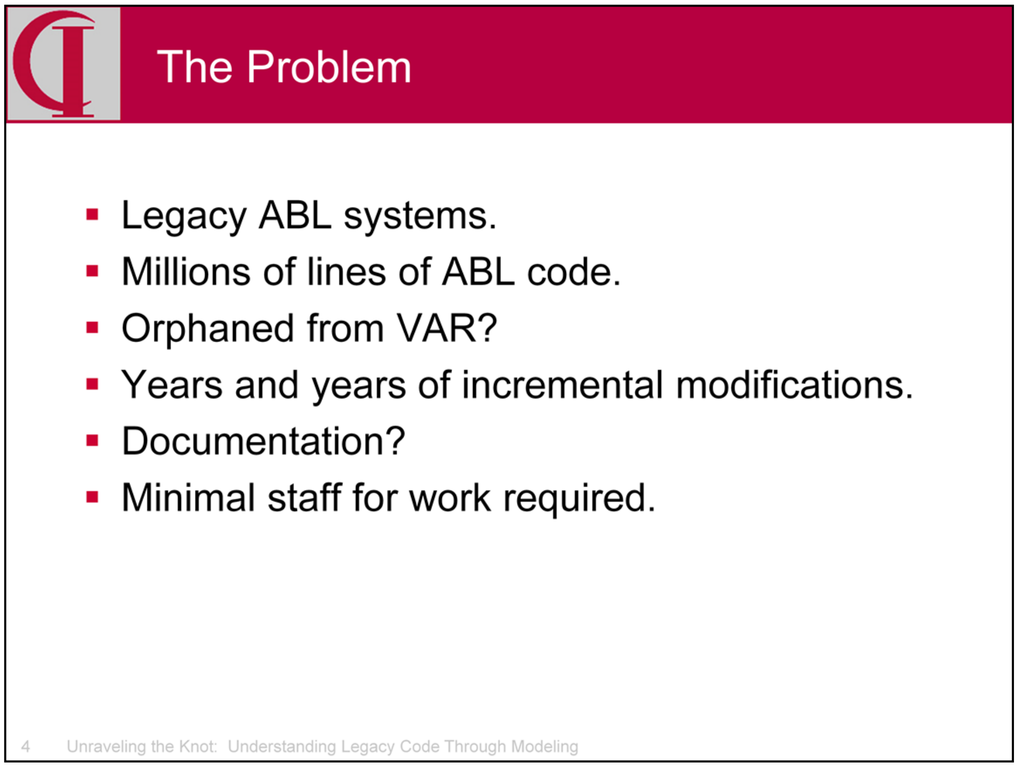


Agenda

- The Problem
- Existing Tools and Their Limitations
- Going Beyond
- UML – What and Why?
- The UML Profile for ABL
- ABL2UML 1.0
- Where Do We Go From Here?
- Summary

3 Unraveling the Knot: Understanding Legacy Code Through Modeling

First, let’s briefly consider what problem we are trying to solve.



The Problem

- Legacy ABL systems.
- Millions of lines of ABL code.
- Orphaned from VAR?
- Years and years of incremental modifications.
- Documentation?
- Minimal staff for work required.

Many ABL systems are 10 or even 20 years old or more.

Many systems were purchased originally from a partner, but

• The partner may no longer be in business; or

• The system is heavily customized so the site cannot upgrade; or

• There may be a breakdown in the relationship with the partner.

The result is an orphaned system.

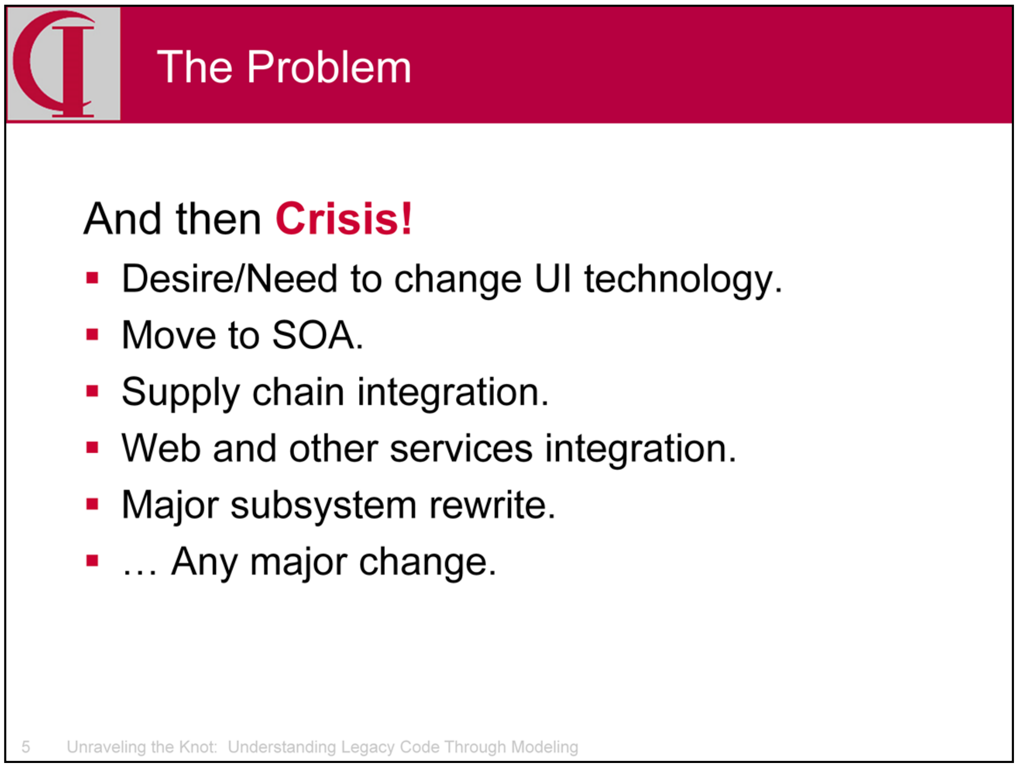
Whether the system was purchased or built entirely in-house, modifications are typically patchwork, on demand, spot changes with no overriding architectural vision.

Progress' low cost of ownership ironically leads to low investment, minimal changes, and limited staff resources.

PSC's principle of upward compatibility means no need for periodic architectural revisions such as are required in most other languages.

Many sites have minimal staff, who just get the work done, with no time or budget for documentation, review, restructuring, etc.

I.e., most shops can go on and on comfortably for many years enjoying the low cost of ownership.



The Problem

And then **Crisis!**

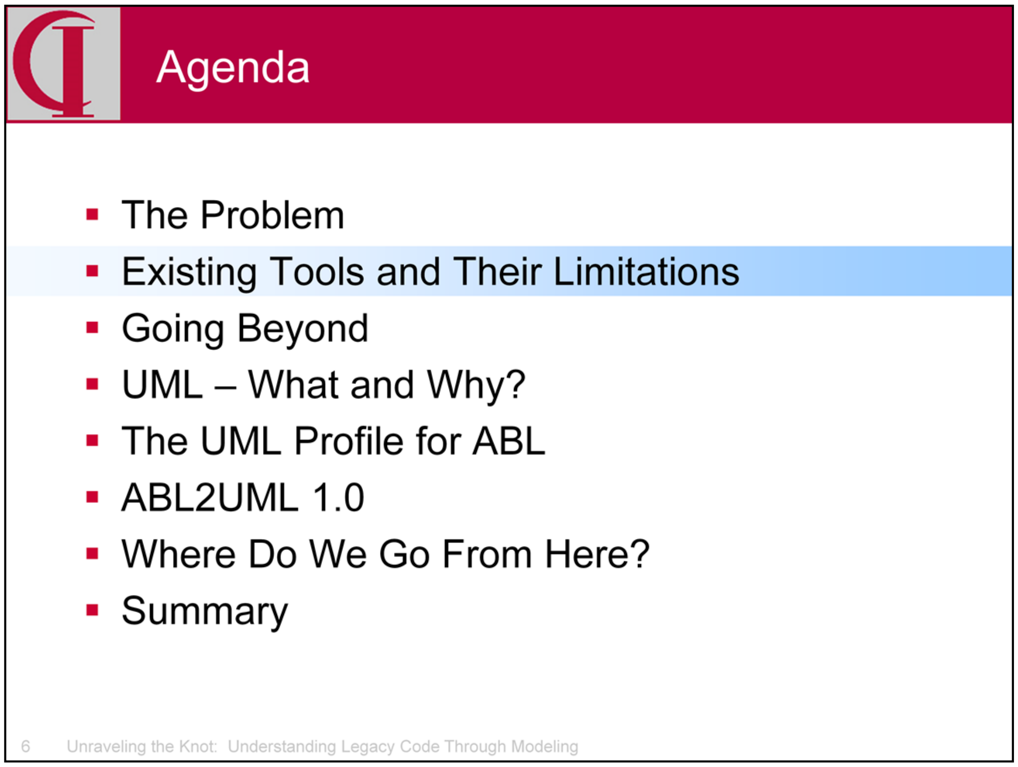
- Desire/Need to change UI technology.
- Move to SOA.
- Supply chain integration.
- Web and other services integration.
- Major subsystem rewrite.
- … Any major change.

Many legacy systems work well and have rich functionality appropriate to the business, but because many older systems are still ChUI, people become dissatisfied with what is seen as old “clunky” interfaces. However, these legacy ChUI architectures typically have UI, BL, and DA all mixed together, making it difficult to just change the UI. This is also true for many older client/server ABL GUI systems. The rewrite to separate UI and introduce a more modern interface can seem like having to nearly rewrite the entire application.

The accumulation of multiple systems in one business can result in near duplication of functions and enormous maintenance burdens. Moving to a SOA, which centralizes each service, is a solution, but a major architectural change.

In the current world, there are many forces driving toward integration – suppliers, customers, and various related services such as shipping, pricing, credit, etc., particularly the increased range and complexity of expectations for web services. But, legacy applications are rarely structured for easy integration and so major architectural changes are required.

Even the need for major changes to a single subsystem can precipitate a crisis because the work is far more extensive than the small modifications which have traditionally been done by the in-house shop.

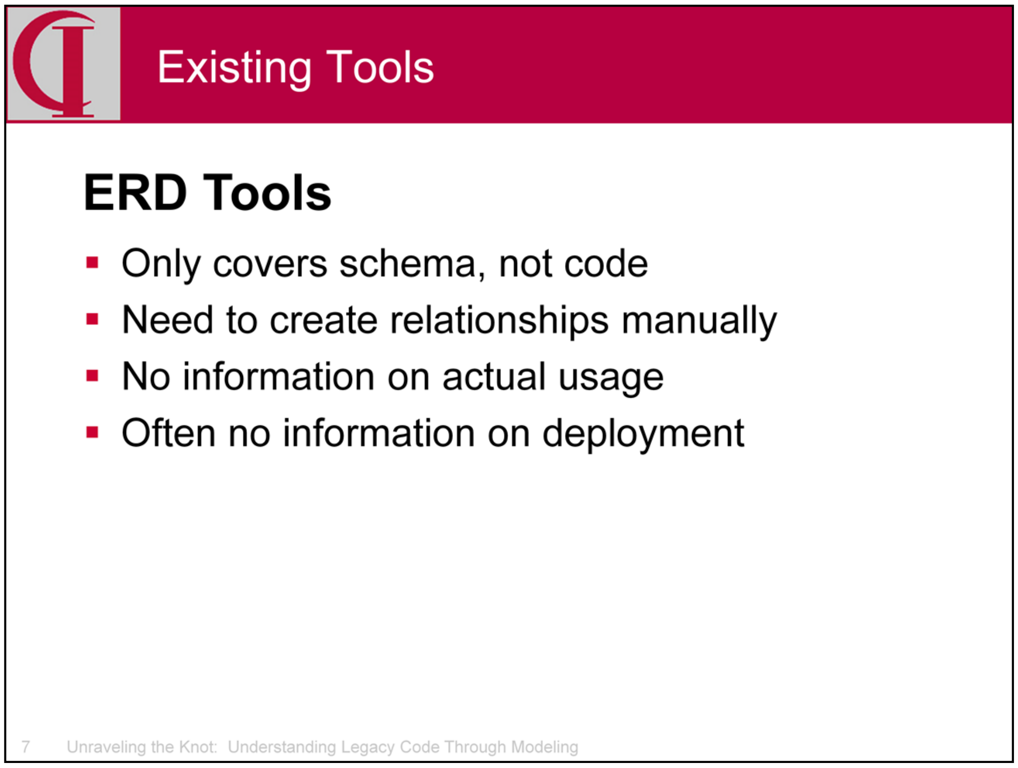


Agenda

- The Problem
- Existing Tools and Their Limitations
- Going Beyond
- UML – What and Why?
- The UML Profile for ABL
- ABL2UML 1.0
- Where Do We Go From Here?
- Summary

6 Unraveling the Knot: Understanding Legacy Code Through Modeling

So, what tools does the Progress community have already?



Existing Tools

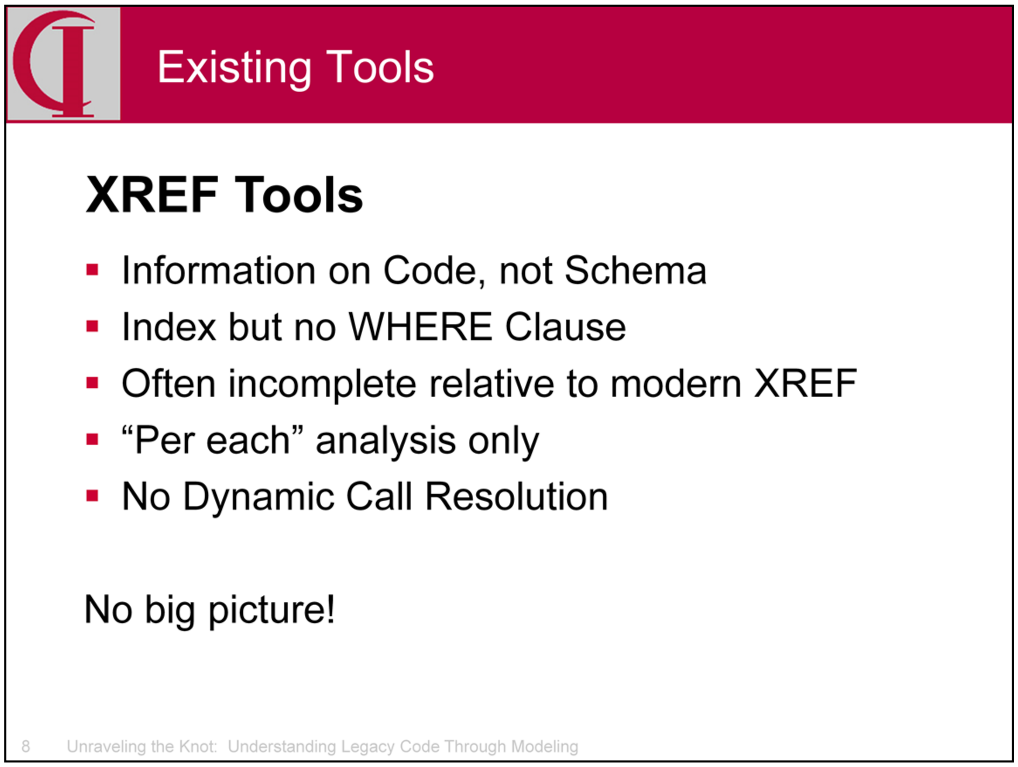
ERD Tools

- Only covers schema, not code
- Need to create relationships manually
- No information on actual usage
- Often no information on deployment

7 Unraveling the Knot: Understanding Legacy Code Through Modeling

Some shops have used ERD (Entity Relationship Diagram) tools to provide some help understanding their applications since few programmers are likely to know and understand the functioning of the hundreds of tables typical of many legacy ABL applications.

Even if the tool supports OpenEdge enough to provide automated initialization from the schema, the schema contains no explicit information about foreign keys. So, relationships between tables have to be created by hand. Moreover, long histories of patchwork development mean there is no guarantee that indexes are currently in use or are good descriptors of how tables are accessed. Individual fields and even entire tables may have fallen into disuse.


Existing Tools

XREF Tools

- Information on Code, not Schema
- Index but no WHERE Clause
- Often incomplete relative to modern XREF
- "Per each" analysis only
- No Dynamic Call Resolution

No big picture!

8 Unraveling the Knot: Understanding Legacy Code Through Modeling

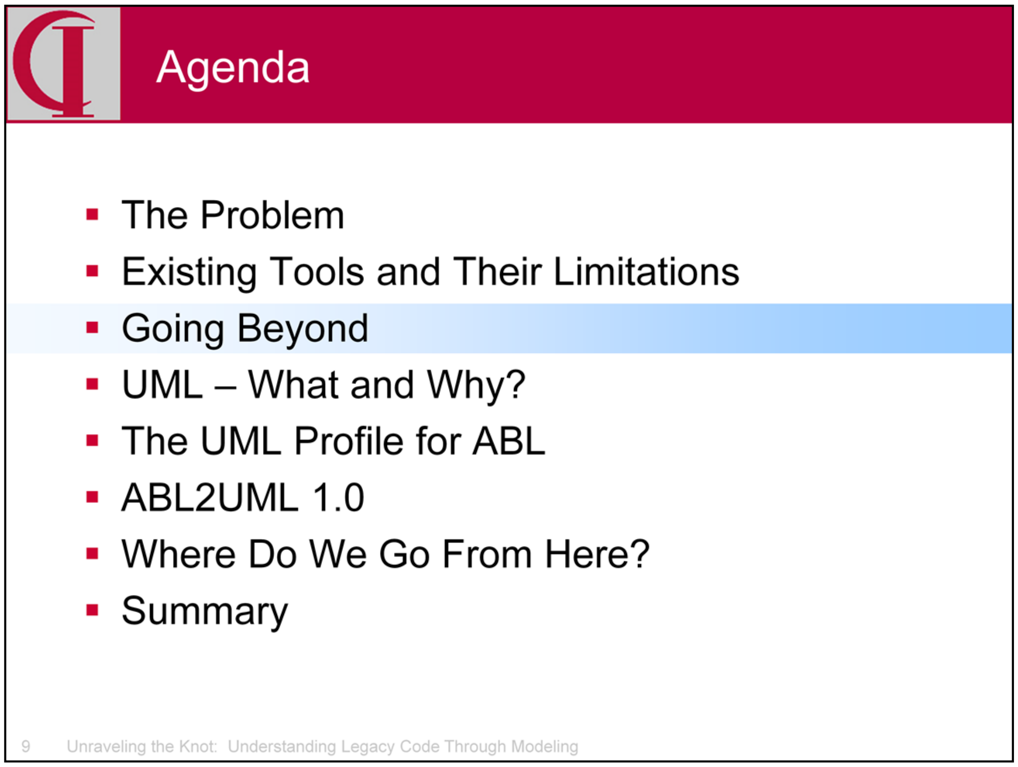
To complement ERD tools, some shops have tools which analyze the output of COMPILE XREF, in some cases storing it in a database for later queries and analysis. This is probably one of the most frequent tool implementations in an ABL shop, although no standard tool has ever emerged to become widely adopted.

This information includes table access and index usage, but not actual WHERE clauses. It has information on what columns within a table are actually being used or modified by the program is imprecise and limited to the containing source file, not the internal procedure or function. This is the kind of information one needs for procedure to service decomposition.

Often, these tools were written years ago and fail to take advantage of additional information now reported by XREF.

There is nothing in the XREF data which helps understand dynamic calls such as RUN VALUE() or a RUN of a procedure in a SUPER, so information on the linkage of program units is very fragmented at best.

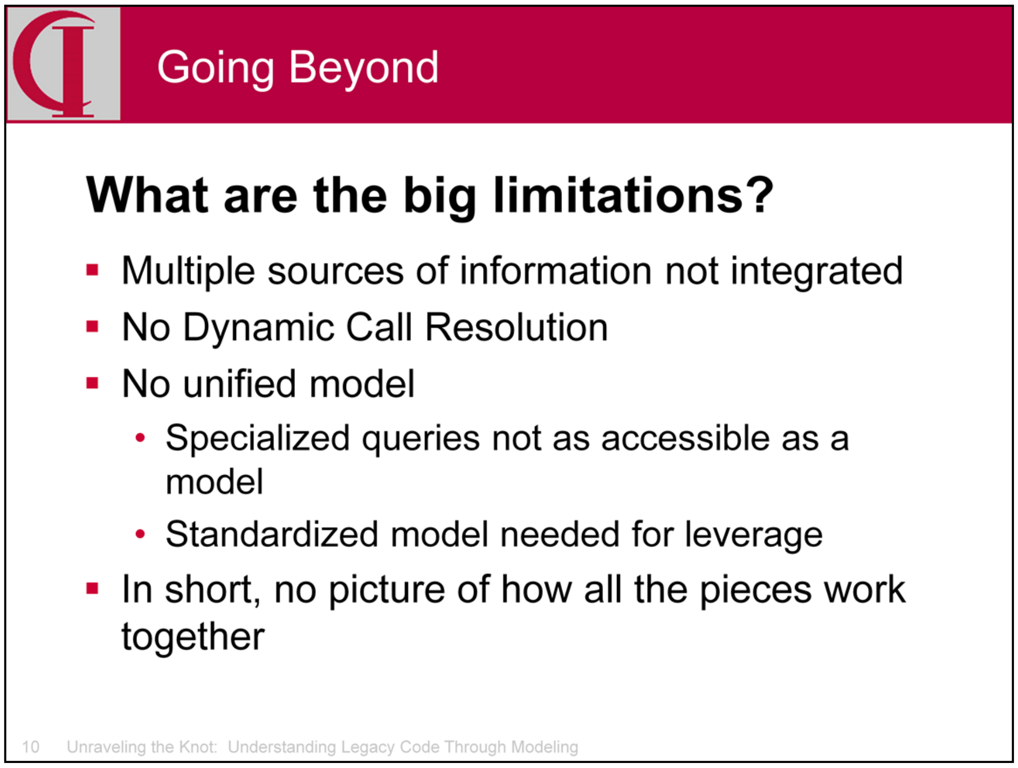
One of the major limitations of these tools is that COMPILE XREF is an analysis of individual compile units, one by one. These tools can be effective in answering questions like "which compile units access table X", but are not useful in describing the operational structure of an application.



Agenda

- The Problem
- Existing Tools and Their Limitations
- Going Beyond
- UML – What and Why?
- The UML Profile for ABL
- ABL2UML 1.0
- Where Do We Go From Here?
- Summary

So, what do we need to go beyond this?



Going Beyond

What are the big limitations?

- Multiple sources of information not integrated
- No Dynamic Call Resolution
- No unified model
  - Specialized queries not as accessible as a model
  - Standardized model needed for leverage
- In short, no picture of how all the pieces work together

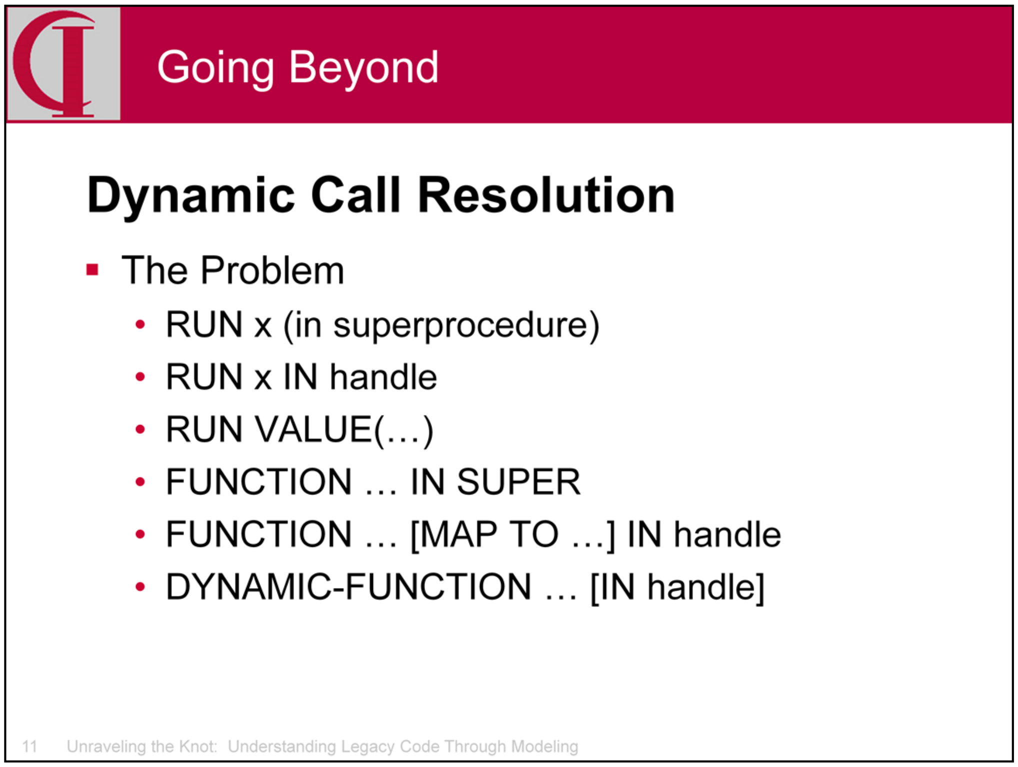
10 Unraveling the Knot: Understanding Legacy Code Through Modeling

While tools like Roundtable may have integrated several tools into a coordinated whole, there is nothing about this integration which transcends the limitations of the individual components.

One of the most conspicuous weaknesses relates to dynamic call resolution, about which I will say more shortly. Any time it is not clear what program, procedure, or function is being executed, the structural relationships in the code are obscured.

Even such information as is available in any given tool set is generally accessible only via limited methods such as queries or browsers, which are far less rich than true modeling systems.

The diversity of tools built shop by shop also means that no leverage has been gained such as would come from many people working on a common standardized modeling tool.


Going Beyond

Dynamic Call Resolution

- The Problem
  - RUN x (in superprocedure)
  - RUN x IN handle
  - RUN VALUE(…)
  - FUNCTION … IN SUPER
  - FUNCTION … [MAP TO …] IN handle
  - DYNAMIC-FUNCTION … [IN handle]

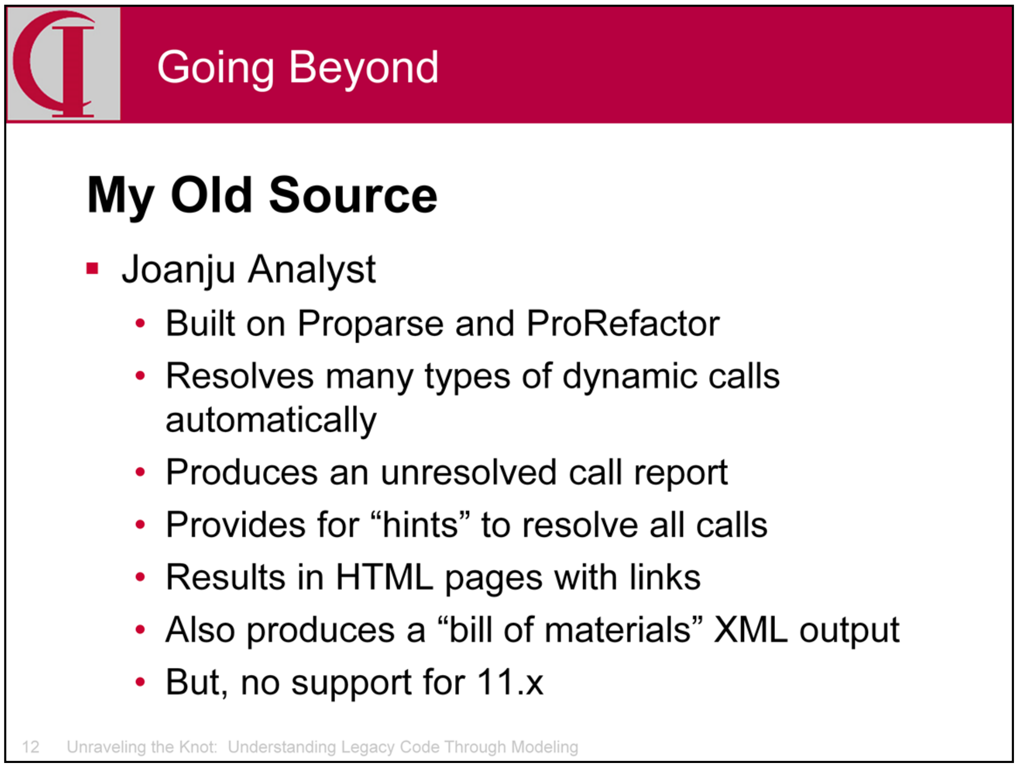
11 Unraveling the Knot: Understanding Legacy Code Through Modeling

Dynamic call mechanisms have added great power to ABL. Even in very early systems, one saw RUN VALUE() with the value coming from a database table, local list, or a computation being heavily used in ABL to provide dynamic operation and simplify what would otherwise have been extensive control logic.

Persistent procedures and later superprocedures have done a great deal to enhance the capabilities of ABL programming, but have also often made it unclear, without reading a lot of code, exactly which code is being executed by which statement. Even when the link between one procedure and a persistent or super procedure is clear, it is not obvious from that context what other procedures might also use that same code.

These same issues exist for function calls.

How severe a problem this is depends on the specific code base, but is likely to be some issue in most.



Going Beyond

My Old Source

- Joanju Analyst
  - Built on Proparse and ProRefactor
  - Resolves many types of dynamic calls automatically
  - Produces an unresolved call report
  - Provides for “hints” to resolve all calls
  - Results in HTML pages with links
  - Also produces a “bill of materials” XML output
  - But, no support for 11.x

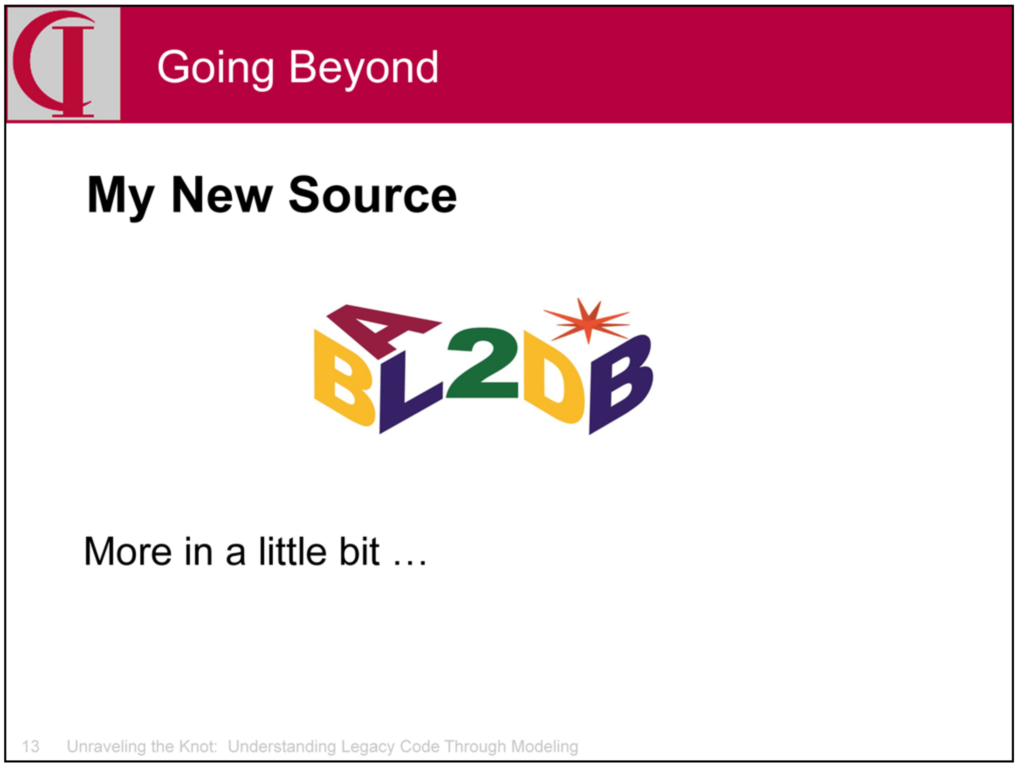
12 Unraveling the Knot: Understanding Legacy Code Through Modeling

Many of you will be familiar with Proparse, which parses ABL code in a fashion similar to what happens during compilation, but which makes the results of that parsing available for analysis by tools such as ProLint, an open source code quality tool originally created by Jurjen Dijkstra. These capabilities were extended in ProRefactor to provide the basic structures and tools for refactoring ABL code. Joanju Software, which created both Proparse and ProRefactor has now built on this foundation to create Analyst, which adds call graph analysis and other features.

Analyst is capable of automatic resolution of many kinds of dynamic calls and provides reporting for those which cannot be resolved automatically. The user can then provide “hints” which will allow Analyst to build a complete call graph of the application. The result is a rich database from which one can dynamically generate HTML pages showing the code with links which allow one to follow the call tree through all its branches. There is also a powerful query tool for inspecting aspects of the structure.

As part of our cooperative effort, John Green of Joanju has provided Analyst with the ability to also produce an XML “bill of materials” file which covers all code components (procedures, classes, internal procedures, functions, etc.), the call path between those units, access to tables, where clauses, and even field-level information on the create, read, update, or delete operations performed by any code component.

Sadly, it has no support for any version past 10.2B. I’ll talk more about this in a few minutes, but let’s see what we accomplished thus far.



Going Beyond

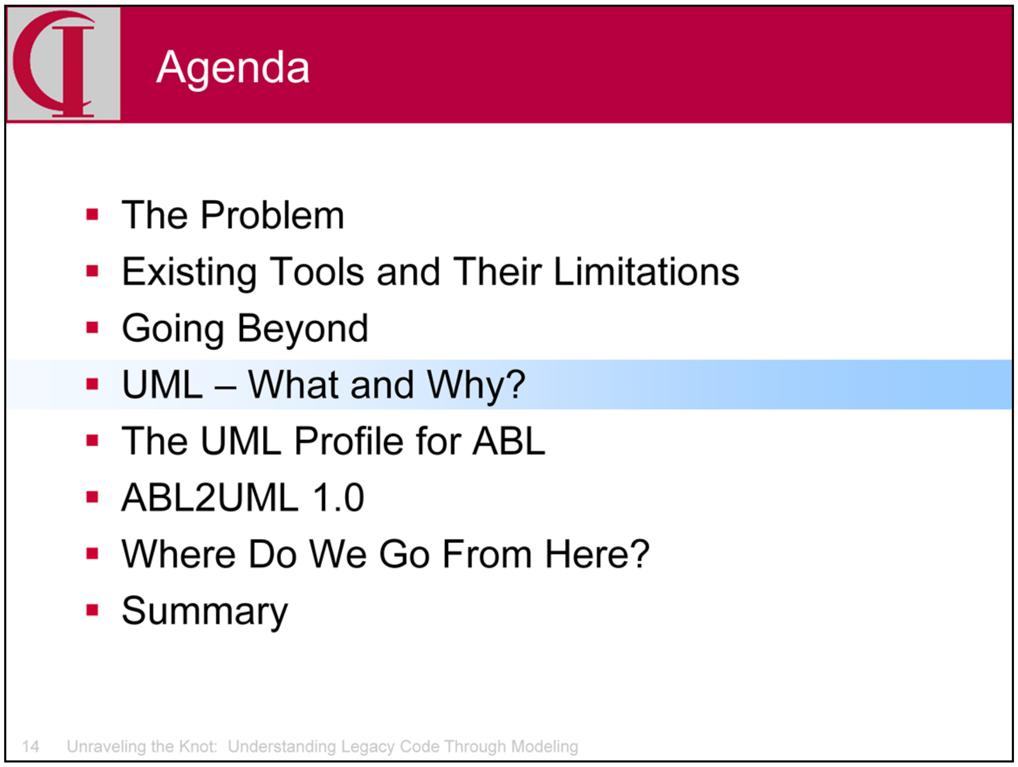
My New Source

ABL2DB

More in a little bit …

13 Unraveling the Knot: Understanding Legacy Code Through Modeling

My proposed replacement for the dependency on Analyst is ABL2DB, which I will talk a bit more about later as well as in some depth at 3:30 tomorrow afternoon.

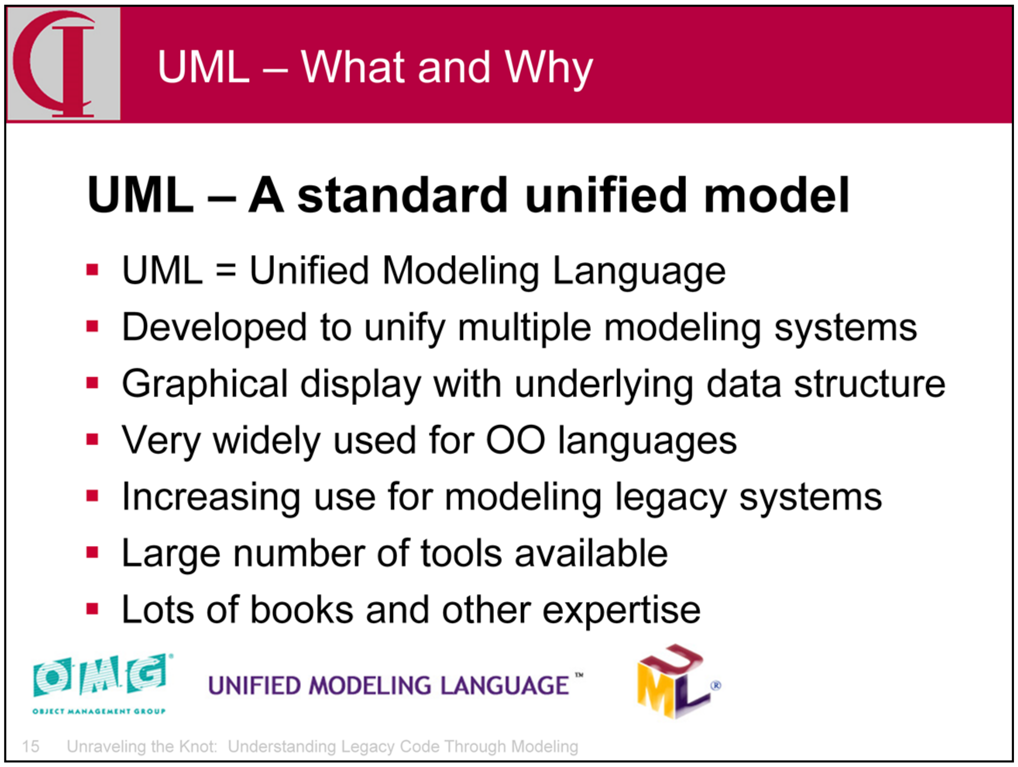


Agenda

- The Problem
- Existing Tools and Their Limitations
- Going Beyond
- UML – What and Why?
- The UML Profile for ABL
- ABL2UML 1.0
- Where Do We Go From Here?
- Summary

14 Unraveling the Knot: Understanding Legacy Code Through Modeling

Given a source of information from Analyst or an alternate tool, let's look at what we were able to accomplish with it. First, we need a way to identify a way to represent the information.



UML – What and Why

UML – A standard unified model

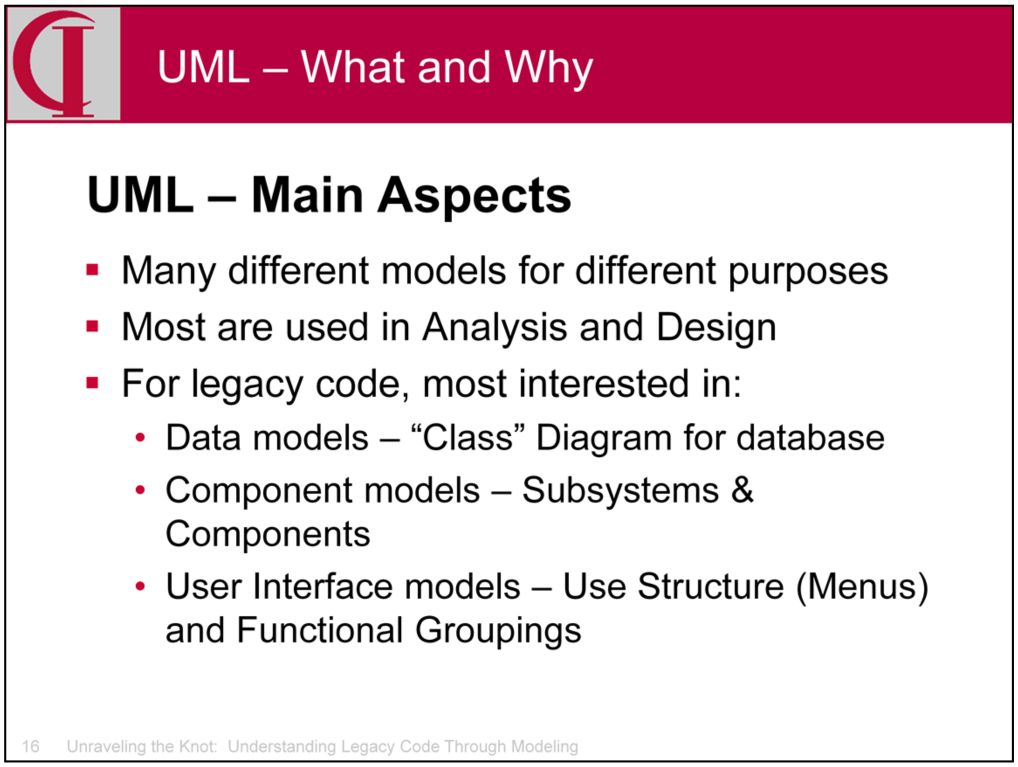
- UML = Unified Modeling Language
- Developed to unify multiple modeling systems
- Graphical display with underlying data structure
- Very widely used for OO languages
- Increasing use for modeling legacy systems
- Large number of tools available
- Lots of books and other expertise

OMG OBJECT MANAGEMENT GROUP UNIFIED MODELING LANGUAGE™ UML®

15 Unraveling the Knot: Understanding Legacy Code Through Modeling

UML was created to unify three different modeling schemes, each of which had its strengths. It represents a rare case in which three competing providers recognized that the absence of a standard weakened all their efforts, so they worked together to create a single superset standard. Since its introduction and broad acceptance, it has grown enormously in scope and capability and is very widely used in object-oriented development. Tools for UML are quite numerous, ranging from free open source efforts to quite expensive proprietary systems, notably Rational.

While its primary use is in designing new applications, reverse engineering of OO applications is quite standard and there is a small but growing, body of work and literature on its use for legacy, non-OO applications, although I confess that I find myself something of an unexpected pioneer in that realm.



UML – What and Why

UML – Main Aspects

- Many different models for different purposes
- Most are used in Analysis and Design
- For legacy code, most interested in:
  - Data models – “Class” Diagram for database
  - Component models – Subsystems & Components
  - User Interface models – Use Structure (Menus) and Functional Groupings

16 Unraveling the Knot: Understanding Legacy Code Through Modeling

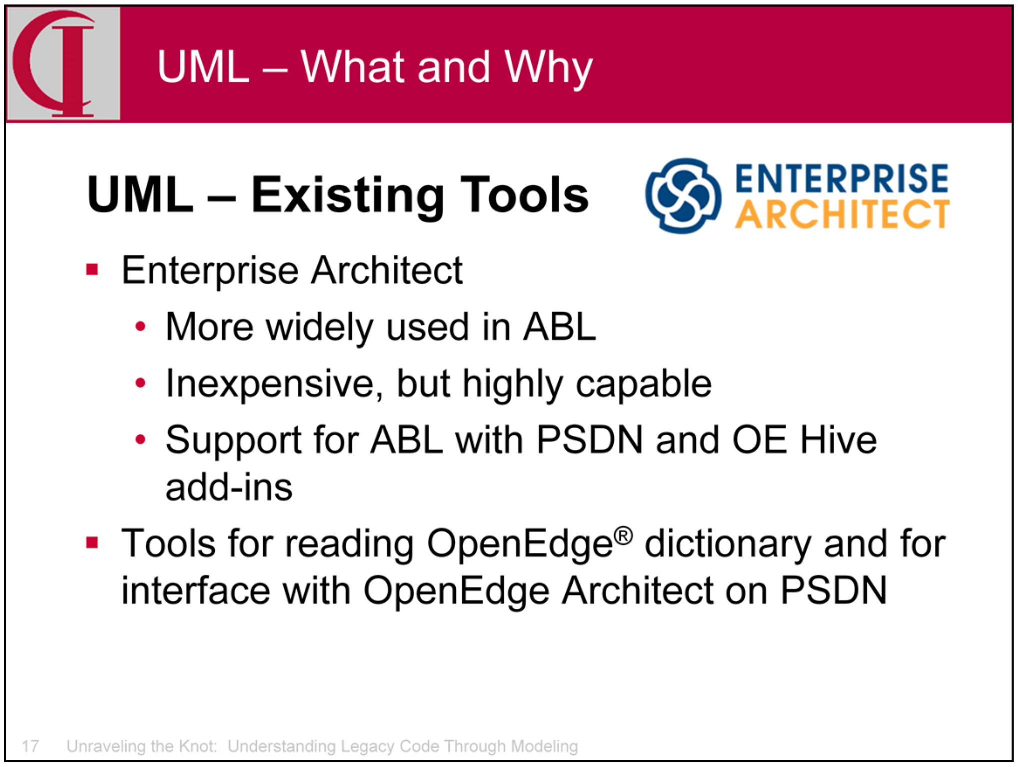
UML 2.0 and 2.1 have become a very rich set of modeling tools. A great many of these are intended for use as part of the Object-Oriented Analysis and Design process which begins with collecting Requirements and Use Cases and proceeds through multiple steps before resulting in the design of a specific system and actual code.

The problem with legacy ABL code, however, is that we have the actual finished system without any of the conceptual design material, and it is that existing system we want to model. Depending on the ultimate purpose, there may also be a need at some point for traditional analytic models, but the first step is to model the system as it has been built. For this, we will primarily use three categories of model – Data Models, Component Models, and User Interface models.

The Data Model is used for all aspects of the OpenEdge schema – databases, tables, columns, indexes, triggers, etc. all properties of those elements.

The Component Model is used for Program Units (.p, .w, and .cls files plus Internal Procedures and Functions), Include Units (.i files), and Compile Units (.r files).

The User Interface Model is used for Menus and Functional Units.



UML – What and Why

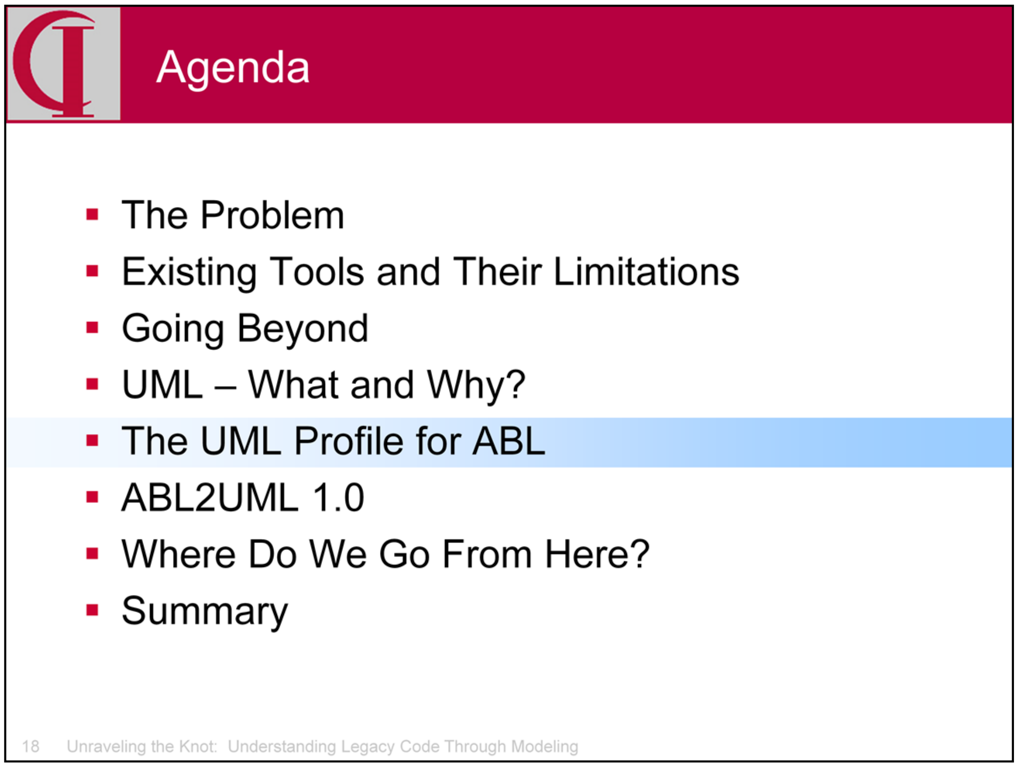
UML – Existing Tools

- Enterprise Architect
  - More widely used in ABL
  - Inexpensive, but highly capable
  - Support for ABL with PSDN and OE Hive add-ins
- Tools for reading OpenEdge® dictionary and for interface with OpenEdge Architect on PSDN

17 Unraveling the Knot: Understanding Legacy Code Through Modeling

While UML is a standard, support for the full standard varies considerably among the available tools, and the tools vary widely in price. In the ABL world, while there are some using Rational (the top of the line, but very expensive), a large number of ABL sites working with UML have adopted Enterprise Architect from Sparx Systems. It offers a very capable tool, very close to Rational in many respects and perhaps even superior in some, but at a very modest price. It supports the use of an OpenEdge database as a repository and while the native product will not read an OpenEdge schema, Phil Magnay of Progress has published a tool on PSDN which will read a .df file and build an EA data model. There have also been some recent publications of his on PSDN for limited reverse engineering in association with OpenEdge Architect.

See https://community.progress.com/technicalusers/w/openedgedevelopment/937.openedge-add-in-for-enterprise-architect.aspx and https://community.progress.com/technicalusers/w/openedgedevelopment/936.openedge-mdg-technology-for-enterprise-architect.aspx on the PSDN Community Wiki pages.

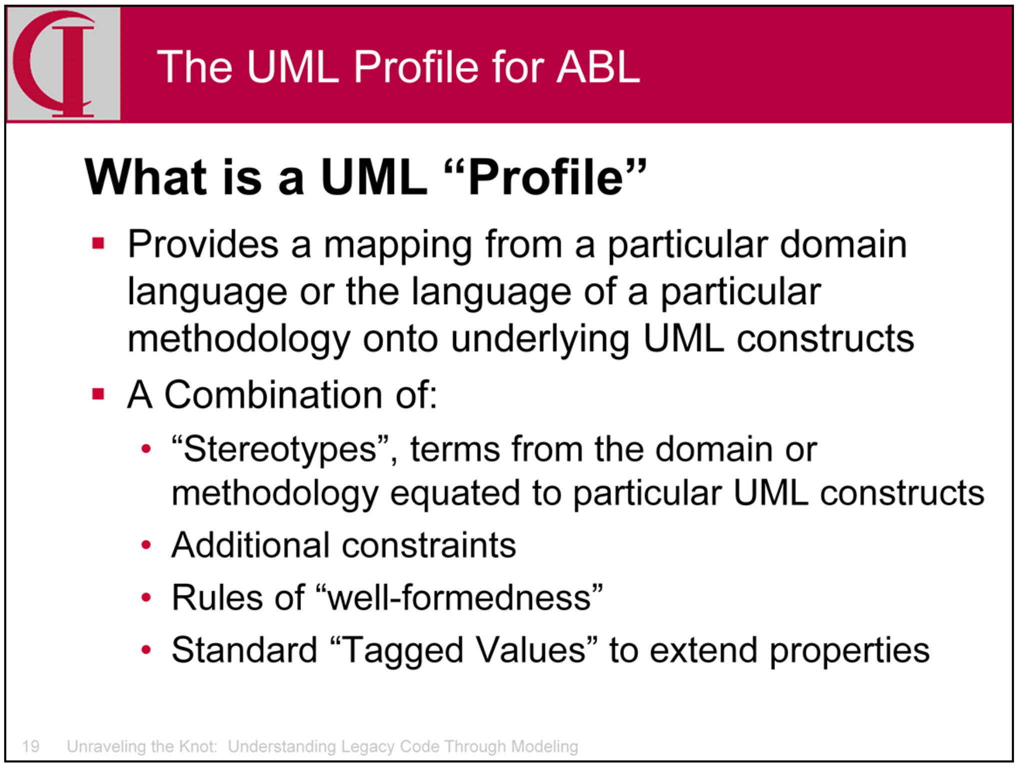


Agenda

- The Problem
- Existing Tools and Their Limitations
- Going Beyond
- UML – What and Why?
- The UML Profile for ABL
- ABL2UML 1.0
- Where Do We Go From Here?
- Summary

18 Unraveling the Knot: Understanding Legacy Code Through Modeling

Having selected UML, we need to do some setup for ABL.

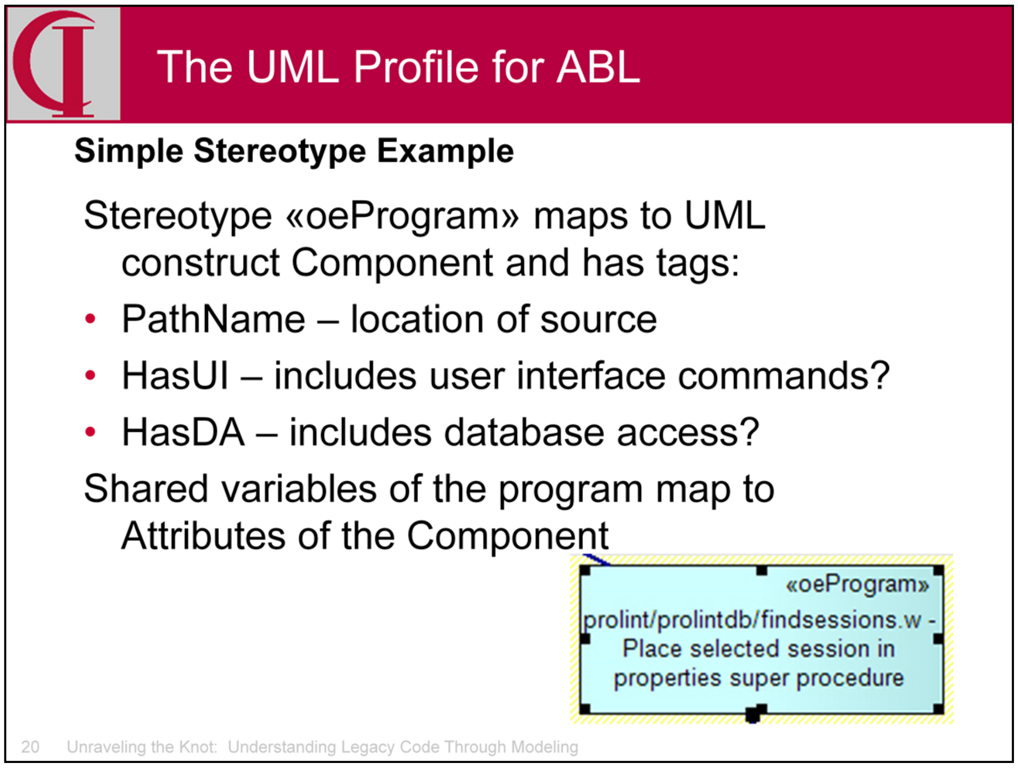


The UML Profile for ABL

What is a UML "Profile"

- Provides a mapping from a particular domain language or the language of a particular methodology onto underlying UML constructs
- A Combination of:
  - "Stereotypes", terms from the domain or methodology equated to particular UML constructs
  - Additional constraints
  - Rules of "well-formedness"
  - Standard "Tagged Values" to extend properties

In UML, there is the concept of a "profile" which is a mechanism for extending the intrinsic parts of UML to provide a vocabulary appropriate to a particular domain or methodology. A profile consists of 'stereotypes' which provide a map between UML elements and connectors to terms appropriate to the domain or methodology. The profile can also include constraints and rules about how the elements are to be used as well as tags which can be used to assign properties to elements beyond those which are inherent to UML. Other standard UML properties include the Alias, often used for a descriptive name, and Notes, which can record more detailed information such as might be extracted from comments, when these are available.



The UML Profile for ABL

Simple Stereotype Example

Stereotype «oeProgram» maps to UML construct Component and has tags:

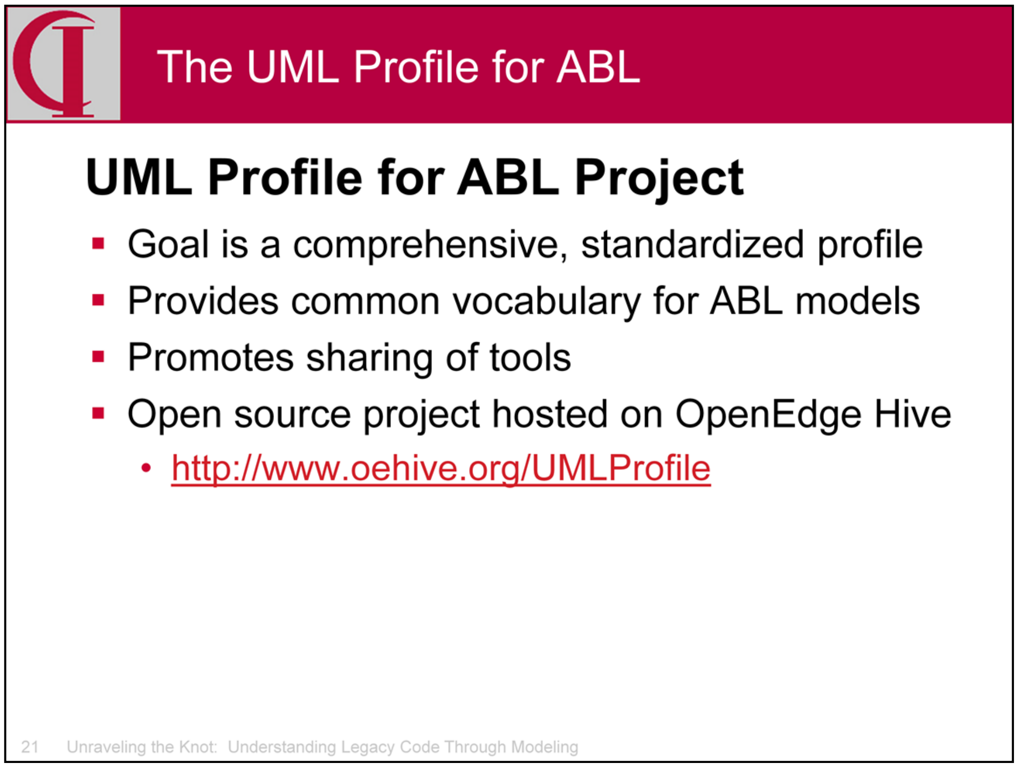
- PathName – location of source
- HasUI – includes user interface commands?
- HasDA – includes database access?

Shared variables of the program map to Attributes of the Component

«oeProgram»
prolint/prolintdb/findsessions.w - Place selected session in properties super procedure

20 Unraveling the Knot: Understanding Legacy Code Through Modeling

As a simple example of what goes into a UML Profile, consider «oeProgram», the stereotype that we use for a .p file. The double brackets called guillemets or angle quotes are used in UML to set off stereotypes. An «oeProgram» is mapped onto the UML construct called Component. Components can have associated Tagged Values and three of these for «oeProgram» are PathName, HasUI, and HasDA. Shared variables in the program are mapped to Attributes of the Component.



The UML Profile for ABL

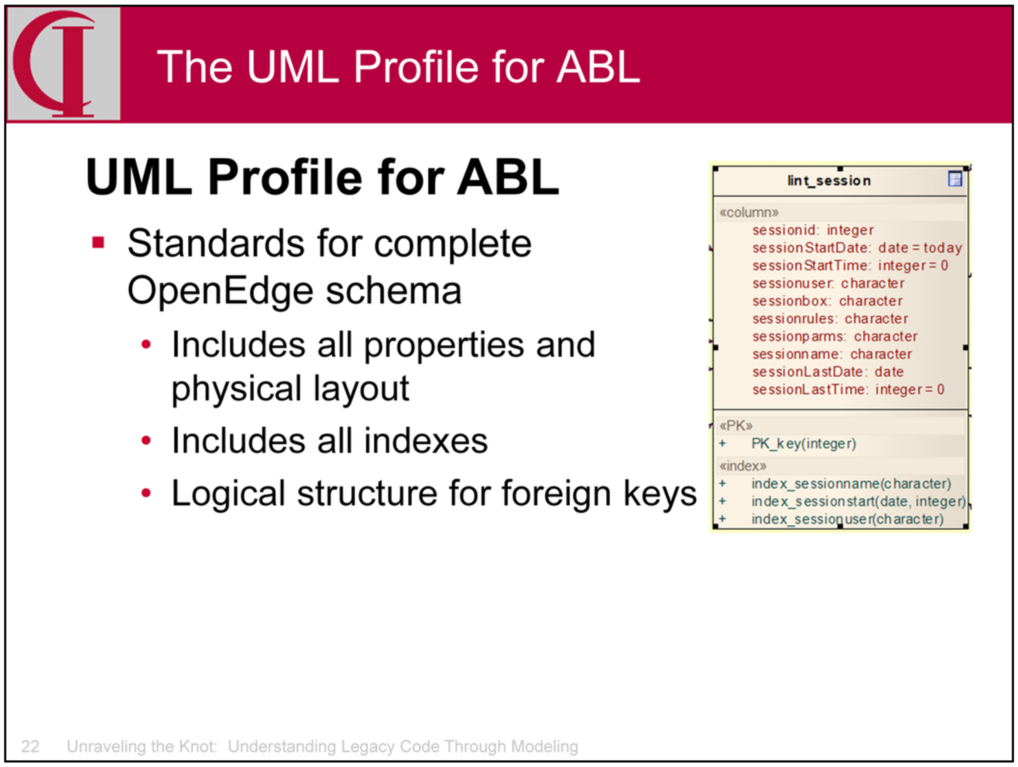
UML Profile for ABL Project

- Goal is a comprehensive, standardized profile
- Provides common vocabulary for ABL models
- Promotes sharing of tools
- Open source project hosted on OpenEdge Hive
  - http://www.oehive.org/UMLProfile

21 Unraveling the Knot: Understanding Legacy Code Through Modeling

Profiles are normally defined for a particular application domain, like telephony, or for a particular form of analysis or methodology. They are not normally used for a language. However, OO languages tend to map onto UML components in a fairly simple way, but ABL, particularly legacy ABL, does not have such an obvious mapping.

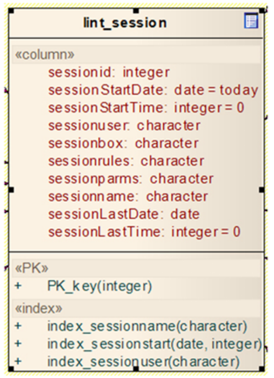
Therefore, I decided that we needed a profile to facilitate modeling of legacy ABL systems. My goal was to create a robust, complete profile which would be suitable for adoption as a standard, thus providing all people working with UML and ABL with a common vocabulary and the potential for sharing of tools. This effort is being treated as an open source project hosted on OpenEdge Hive. I encourage others working with ABL and UML to contribute to and adopt this profile.



The UML Profile for ABL

UML Profile for ABL

- Standards for complete OpenEdge schema
  - Includes all properties and physical layout
  - Includes all indexes
  - Logical structure for foreign keys

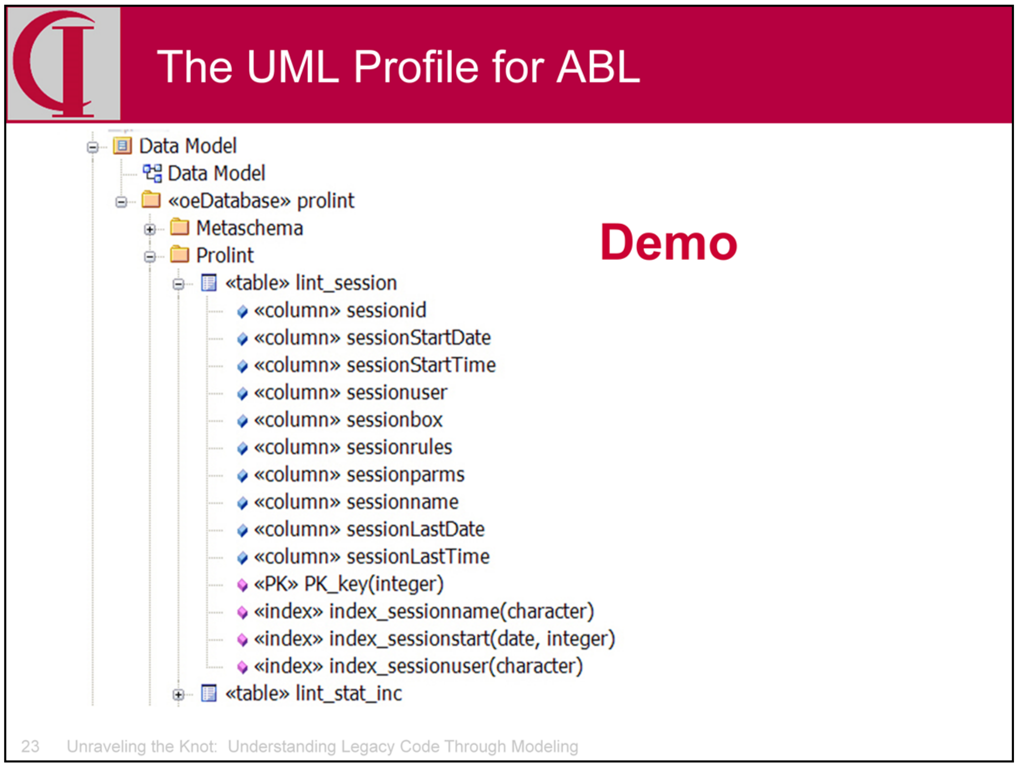


```
lint_session
«column»
      sessionid: integer
      sessionStartDate: date = today
      sessionStartTime: integer = 0
      sessionuser: character
      sessionbox: character
      sessionrules: character
      sessionparms: character
      sessionname: character
      sessionLastDate: date
      sessionLastTime: integer = 0
«PK»
+     PK_key(integer)
«index»
+     index_sessionname(character)
+     index_sessionstart(date, integer)
+     index_sessionuser(character)
```

22 Unraveling the Knot: Understanding Legacy Code Through Modeling

The stereotypes and tags for the data model cover all properties in the schema, including physical layout of the database, although we have not yet implemented the DataServer portions.

The tools Phil Magnay published include one that will “discover” foreign key relationships by looking for field name matches. This might be helpful for some databases, but fails on those which use table-specific field names or those which have common auditing fields in many tables, resulting in false keys. A structure for keys and relationships has been included in the profile, but I have not provided code for this in the initial ABL to UML tool. However, the tool does show actual use.



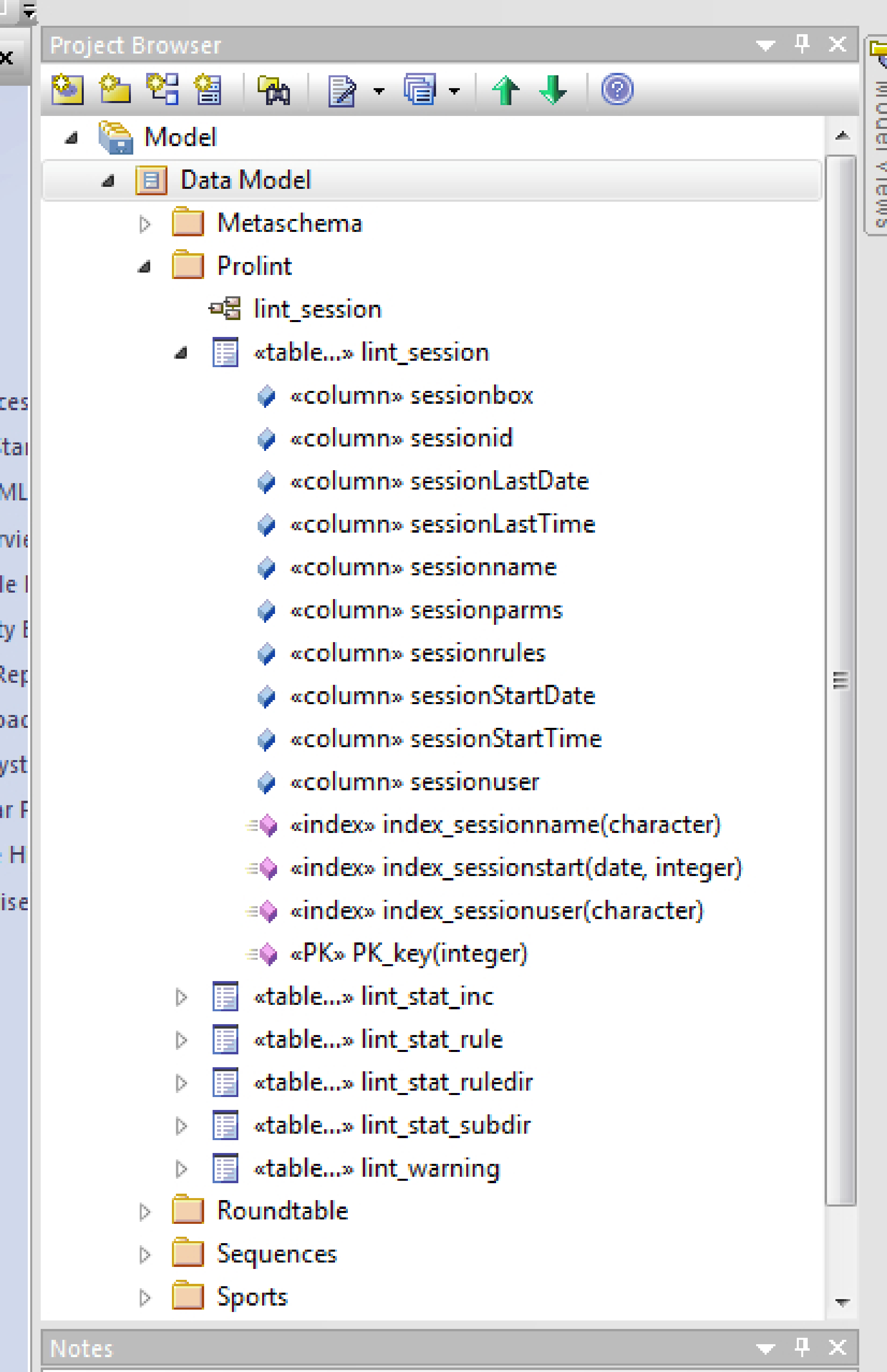
The UML Profile for ABL

```
Data Model
  Data Model
  «oeDatabase» prolint
    Metaschema
    Prolint
      «table» lint_session
        «column» sessionid
        «column» sessionStartDate
        «column» sessionStartTime
        «column» sessionuser
        «column» sessionbox
        «column» sessionrules
        «column» sessionparms
        «column» sessionname
        «column» sessionLastDate
        «column» sessionLastTime
        «PK» PK_key(integer)
        «index» index_sessionname(character)
        «index» index_sessionstart(date, integer)
        «index» index_sessionuser(character)
      «table» lint_stat_inc
```

Demo

23 Unraveling the Knot: Understanding Legacy Code Through Modeling

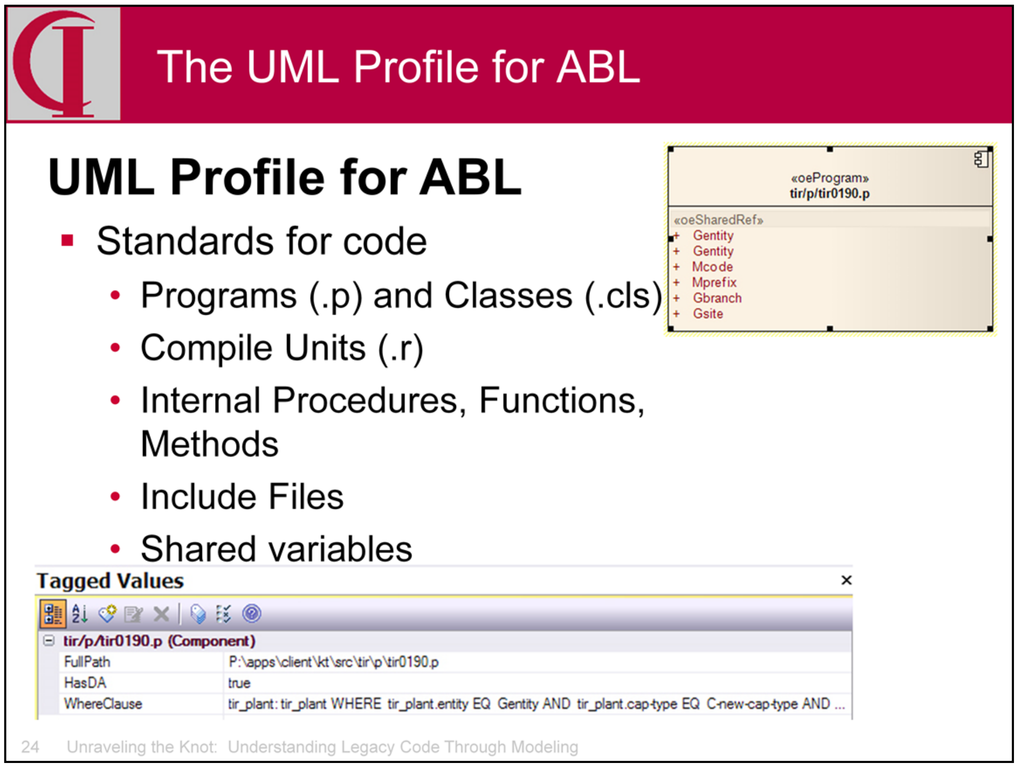
Here is a quick look at a piece of the Data Model for ProLint as it is displayed in the Project Browser panel of Enterprise Architect. As noted, all UML examples in this presentation are taken from an analysis of ProLint. Here one can see that a single database has been divided into packages for Metaschema and the ProLint schema proper. Packages for pieces of the Roundtable and Sports schema also exist, but do not show here. One can see a table with its columns, primary key, and indexes. Of course, there are a large number of details which don't show in this view and which are available by looking at the details of the table object.



Project Browser

- Model
  - Data Model
    - Metaschema
    - Prolint
      - lint_session
      - «table...» lint_session
        - «column» sessionbox
        - «column» sessionid
        - «column» sessionLastDate
        - «column» sessionLastTime
        - «column» sessionname
        - «column» sessionparms
        - «column» sessionrules
        - «column» sessionStartDate
        - «column» sessionStartTime
        - «column» sessionuser
        - «index» index_sessionname(character)
        - «index» index_sessionstart(date, integer)
        - «index» index_sessionuser(character)
        - «PK» PK_key(integer)
      - «table...» lint_stat_inc
      - «table...» lint_stat_rule
      - «table...» lint_stat_ruledir
      - «table...» lint_stat_subdir
      - «table...» lint_warning
    - Roundtable
    - Sequences
    - Sports

Notes



The UML Profile for ABL

UML Profile for ABL

- Standards for code
  - Programs (.p) and Classes (.cls)
  - Compile Units (.r)
  - Internal Procedures, Functions, Methods
  - Include Files
  - Shared variables

«oeProgram»
tir/p/tir0190.p

«oeSharedRef»
+ Gentity
+ Gentity
+ Mcode
+ Mprefix
+ Gbranch
+ Gsite

Tagged Values

tir/p/tir0190.p (Component)

FullPath	P:\apps\client\kt\src\tir\p\tir0190.p
HasDA	true
WhereClause	tir_plant: tir_plant WHERE tir_plant.entity EQ Gentity AND tir_plant.cap-type EQ C-new-cap-type AND ...

24 Unraveling the Knot: Understanding Legacy Code Through Modeling

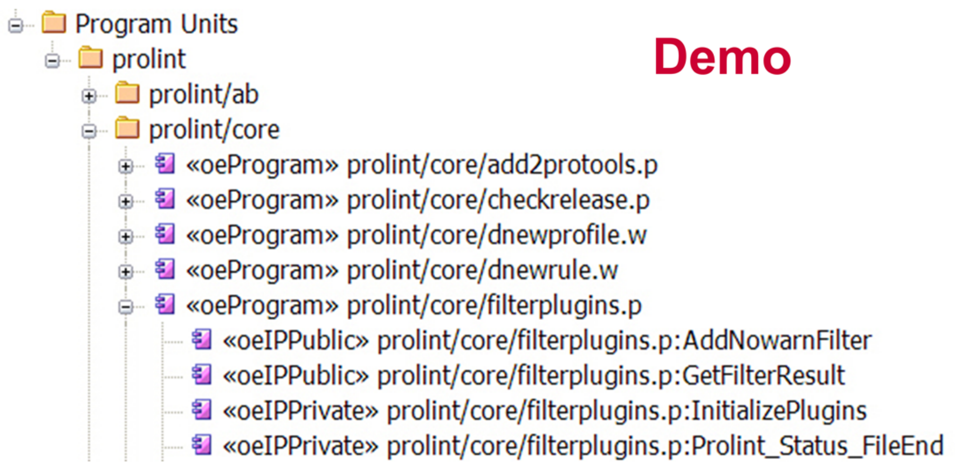
The stereotypes for code emphasize the structural units of the code, i.e., those subdivisions of the code from which and to which control flows. Thus, there are stereotypes for programs, classes, and compile units at the top level, and internal procedures, functions, and methods for the internal structures. Include files are treated separately since they may be used in multiple places.

As a part of the analysis in building the model, internal procedures and functions are classified as public or private based on whether any of the references to them come from outside the compile unit.

All new shared definitions and references to shared variables are also included in the Profile.

As tools evolve, I hope to include more aspects of the code as additional stereotypes and tags. This will depend largely on what the market needs.

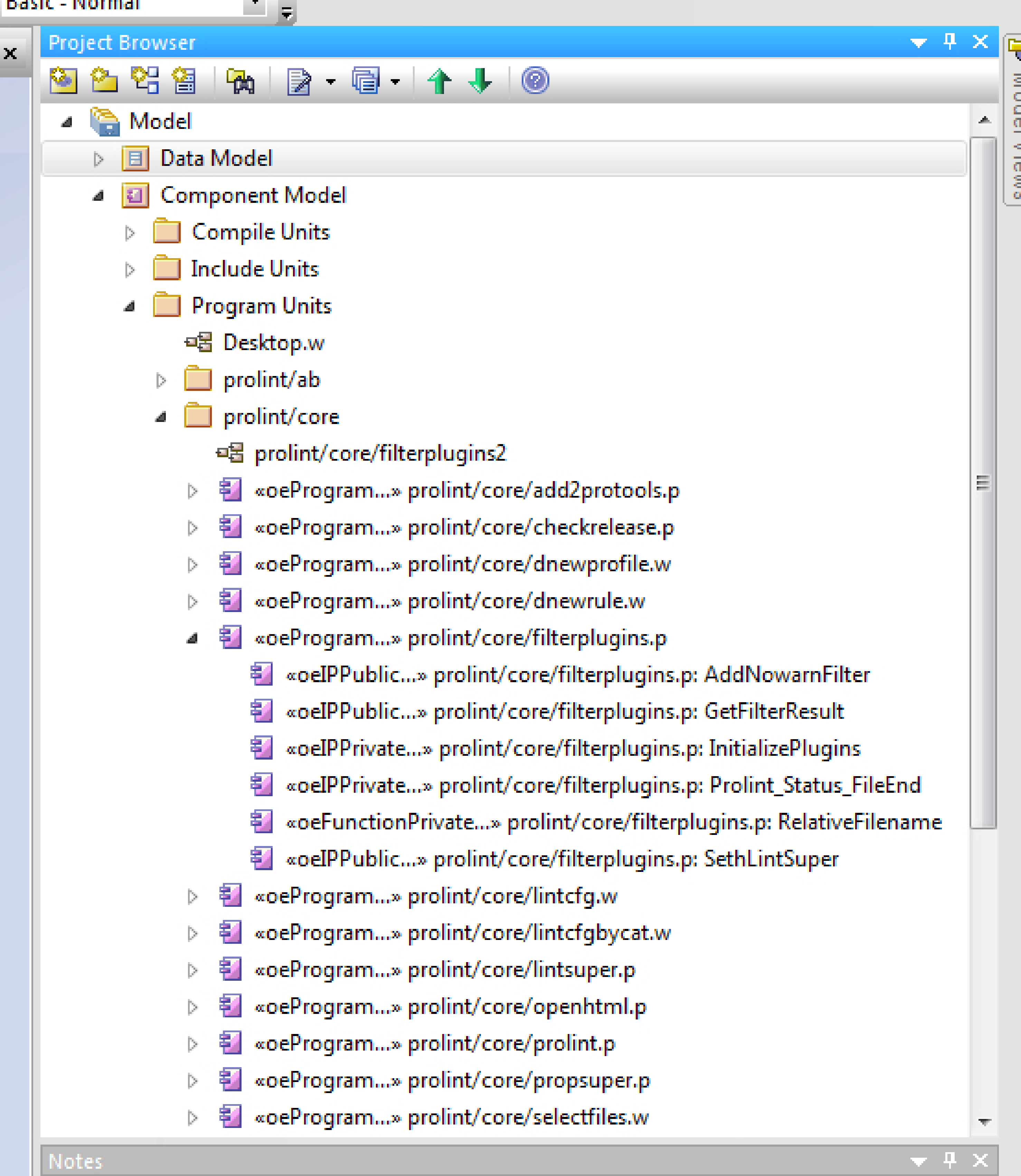
The UML Profile for ABL



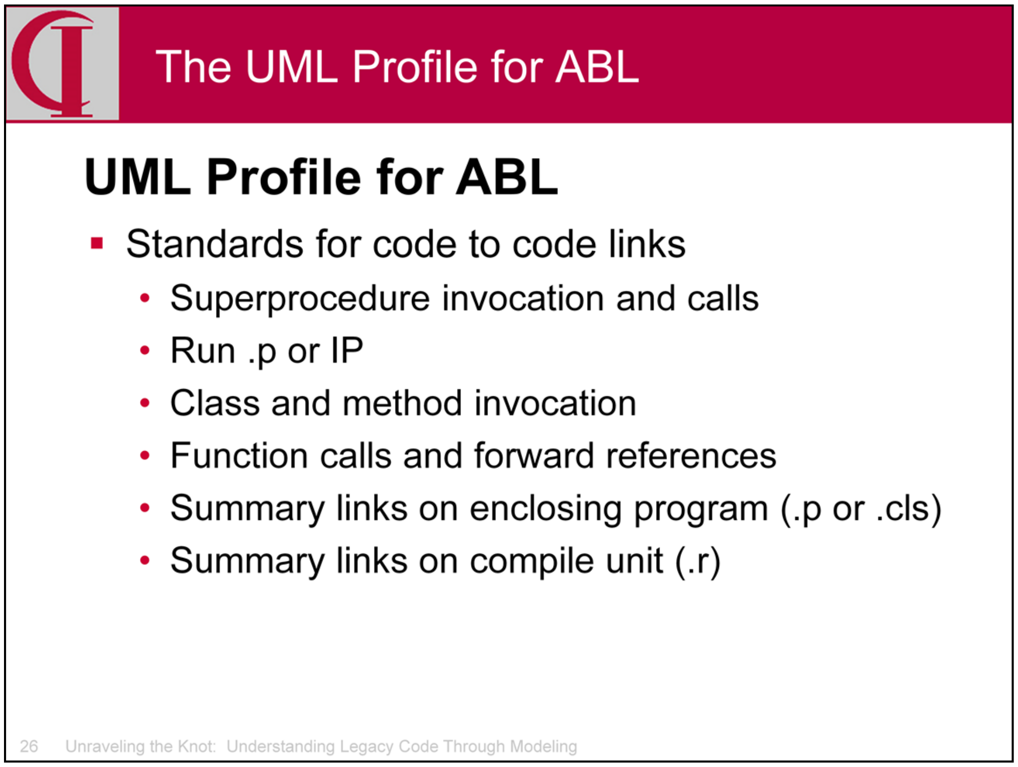
Demo

- Program Units
  - prolint
    - prolint/ab
    - prolint/core
      - «oeProgram» prolint/core/add2protools.p
      - «oeProgram» prolint/core/checkrelease.p
      - «oeProgram» prolint/core/dnewprofile.w
      - «oeProgram» prolint/core/dnewrule.w
      - «oeProgram» prolint/core/filterplugins.p
        - «oeIPPublic» prolint/core/filterplugins.p:AddNowarnFilter
        - «oeIPPublic» prolint/core/filterplugins.p:GetFilterResult
        - «oeIPPrivate» prolint/core/filterplugins.p:InitializePlugins
        - «oeIPPrivate» prolint/core/filterplugins.p:Prolint_Status_FileEnd

Within the Component Model there are packages for Compile Units, Include Units, and Program Units. Compile Units cover all compiled code packages, i.e. .r file. Include Units cover all .i files. Program Units cover all source except Include Units. A portion of the Program Unit hierarchy from ProLint is shown here, where you can see both private and public IPs under the filterplugins.p program.



Model
Data Model
Component Model
Compile Units
Include Units
Program Units
Desktop.w
prolint/ab
prolint/core
prolint/core/filterplugins2
«oeProgram...» prolint/core/add2protools.p
«oeProgram...» prolint/core/checkrelease.p
«oeProgram...» prolint/core/dnewprofile.w
«oeProgram...» prolint/core/dnewrule.w
«oeProgram...» prolint/core/filterplugins.p
«oeIPPublic...» prolint/core/filterplugins.p: AddNowarnFilter
«oeIPPublic...» prolint/core/filterplugins.p: GetFilterResult
«oeIPPrivate...» prolint/core/filterplugins.p: InitializePlugins
«oeIPPrivate...» prolint/core/filterplugins.p: Prolint_Status_FileEnd
«oeFunctionPrivate...» prolint/core/filterplugins.p: RelativeFilename
«oeIPPublic...» prolint/core/filterplugins.p: SethLintSuper
«oeProgram...» prolint/core/lintcfg.w
«oeProgram...» prolint/core/lintcfgbycat.w
«oeProgram...» prolint/core/lintsuper.p
«oeProgram...» prolint/core/openhtml.p
«oeProgram...» prolint/core/prolint.p
«oeProgram...» prolint/core/propsuper.p
«oeProgram...» prolint/core/selectfiles.w



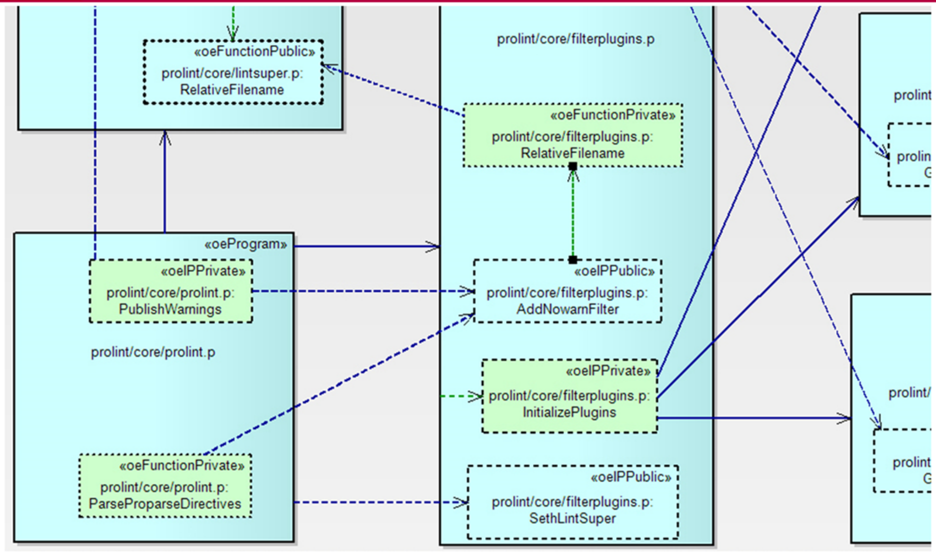
The UML Profile for ABL

UML Profile for ABL

- Standards for code to code links
  - Superprocedure invocation and calls
  - Run .p or IP
  - Class and method invocation
  - Function calls and forward references
  - Summary links on enclosing program (.p or .cls)
  - Summary links on compile unit (.r)

Stereotypes for control flow connections between code components cover all types of possible relationships including simple run statements of programs or internal procedures, new class statements, method invocations, function calls and forward references, and the add of local or session superprocedures. For all control flow links, there is both a detail link between the actual code units in which the flow begins and end and there is a summary link between the enclosing procedures or classes. A second summary link is created between the compile units. This facilitates diagramming at multiple levels.

The UML Profile for ABL



Demo

Here you can see a portion of a UML diagram illustrating a portion of ProLint.

• Blue boxes with solid boundaries are programs;

• Dashed boundaries are internal procedures;

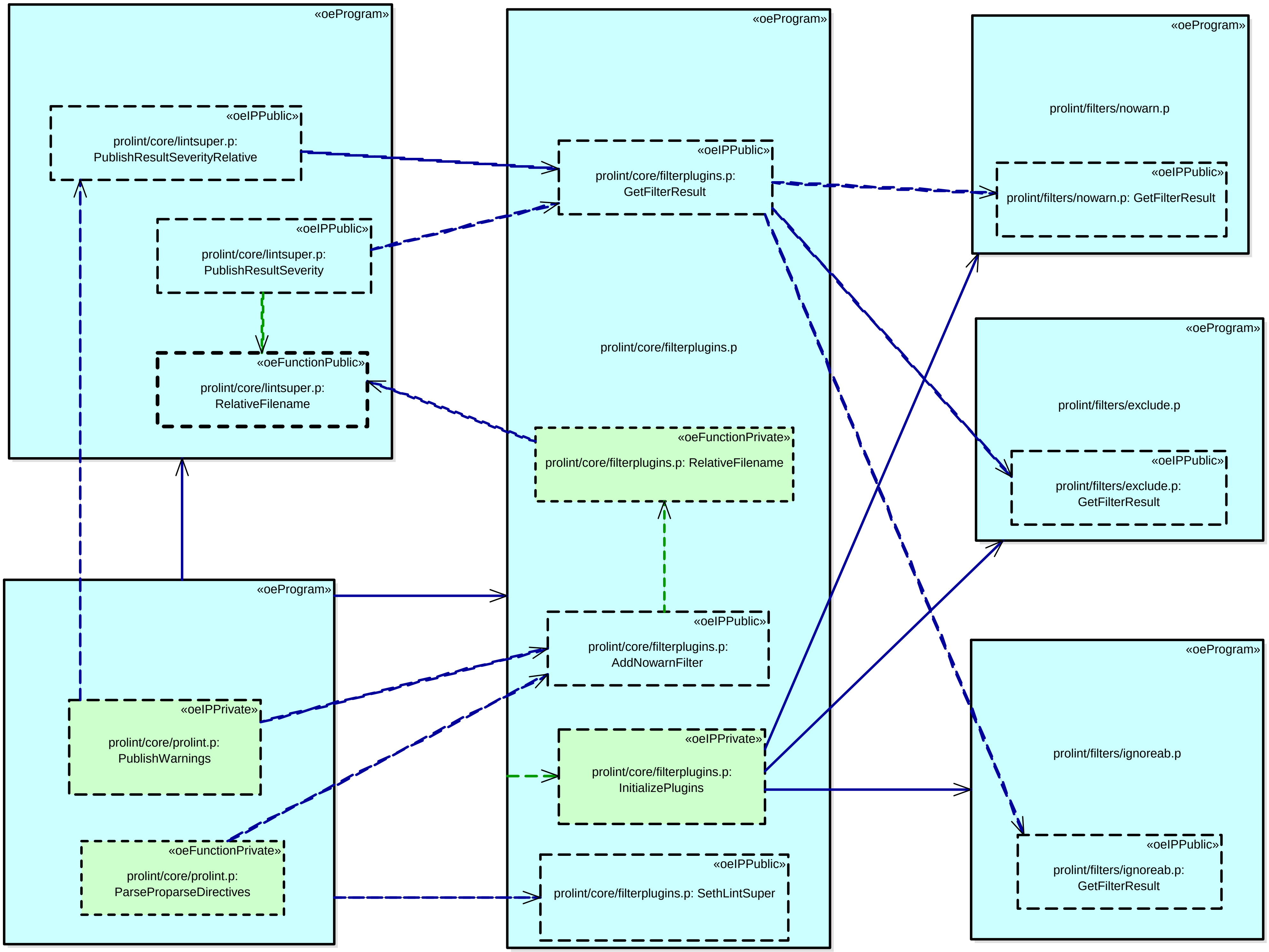
• Dotted boundaries are functions;

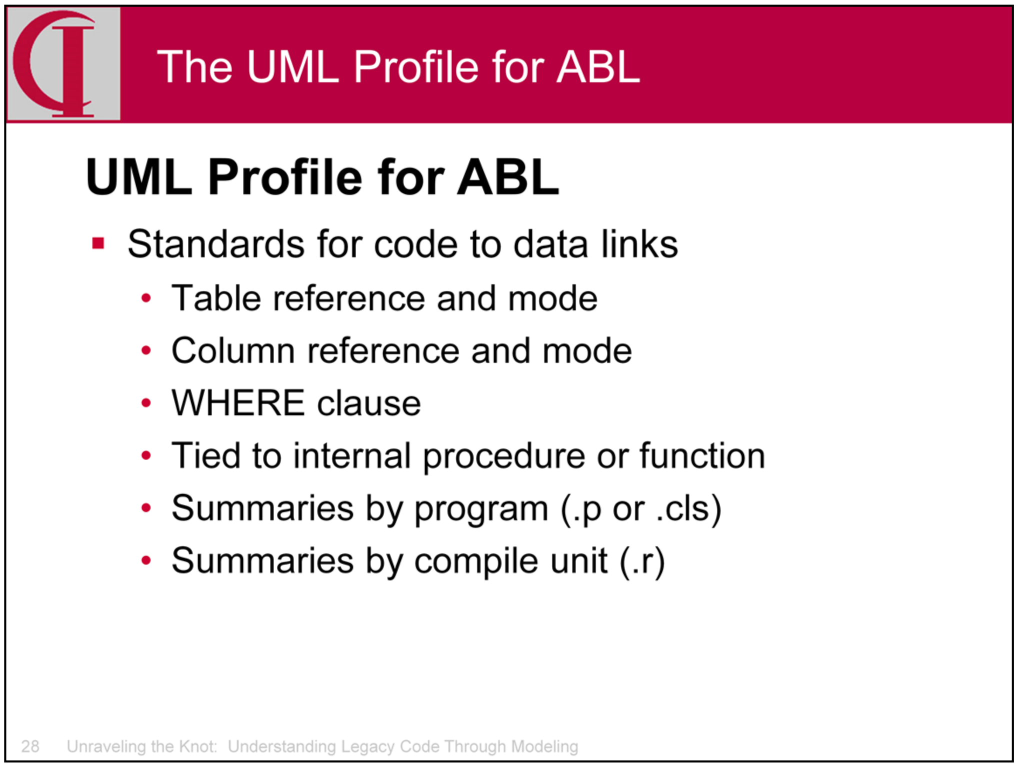
• Blue IPs and Functions are public; and

• Green IPs and Functions are private.

Similarly, run program, run IP, and function calls are solid, dashed, dotted, and colored correspondingly.

The standard stereotype definitions available on OpenEdge Hive include the scripts to provide this coloration and line treatment in Enterprise Architect. A key is available there describing the standards used.


«oeProgram»
«oeIPPublic»
prolint/core/lintsuper.p: PublishResultSeverityRelative
«oeIPPublic»
prolint/core/lintsuper.p: PublishResultSeverity
«oeFunctionPublic»
prolint/core/lintsuper.p: RelativeFilename
«oeProgram»
«oeIPPrivate»
prolint/core/prolint.p: PublishWarnings
«oeFunctionPrivate»
prolint/core/prolint.p: ParseProparseDirectives
«oeProgram»
«oeIPPublic»
prolint/core/filterplugins.p: GetFilterResult
prolint/core/filterplugins.p
«oeFunctionPrivate»
prolint/core/filterplugins.p: RelativeFilename
«oeIPPublic»
prolint/core/filterplugins.p: AddNowarnFilter
«oeIPPrivate»
prolint/core/filterplugins.p: InitializePlugins
«oeIPPublic»
prolint/core/filterplugins.p: SethLintSuper
«oeProgram»
prolint/filters/nowarn.p
«oeIPPublic»
prolint/filters/nowarn.p: GetFilterResult
«oeProgram»
prolint/filters/exclude.p
«oeIPPublic»
prolint/filters/exclude.p: GetFilterResult
«oeProgram»
prolint/filters/ignoreab.p
«oeIPPublic»
prolint/filters/ignoreab.p: GetFilterResult


The UML Profile for ABL

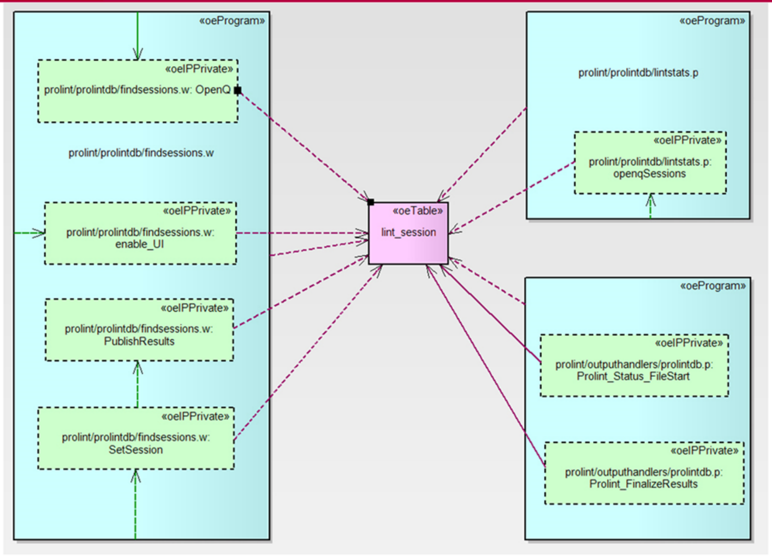
UML Profile for ABL

- Standards for code to data links
  - Table reference and mode
  - Column reference and mode
  - WHERE clause
  - Tied to internal procedure or function
  - Summaries by program (.p or .cls)
  - Summaries by compile unit (.r)

28 Unraveling the Knot: Understanding Legacy Code Through Modeling

Detail links are created between the smallest containing source code unit, e.g., program, class, internal procedure, function, or method, and are summarized at the level of the containing program and compile unit. Stereotypes distinguish between read only links and those where create, delete, or update activity occur and the type of activity is provided in a tag. The actual WHERE clause for any query is also captured. Each field which is explicitly read or written is noted in a tag along with the action which occurs.

The UML Profile for ABL

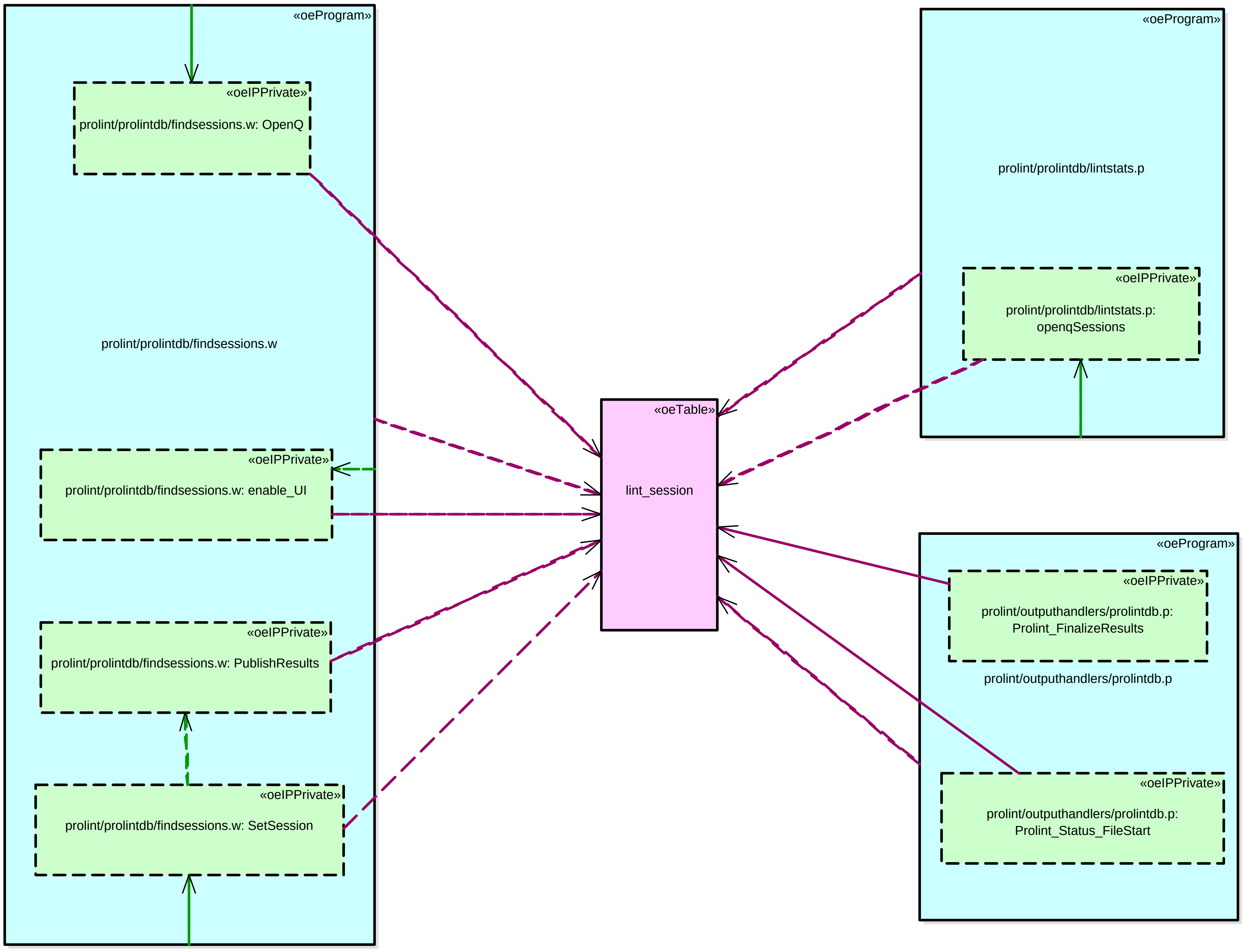


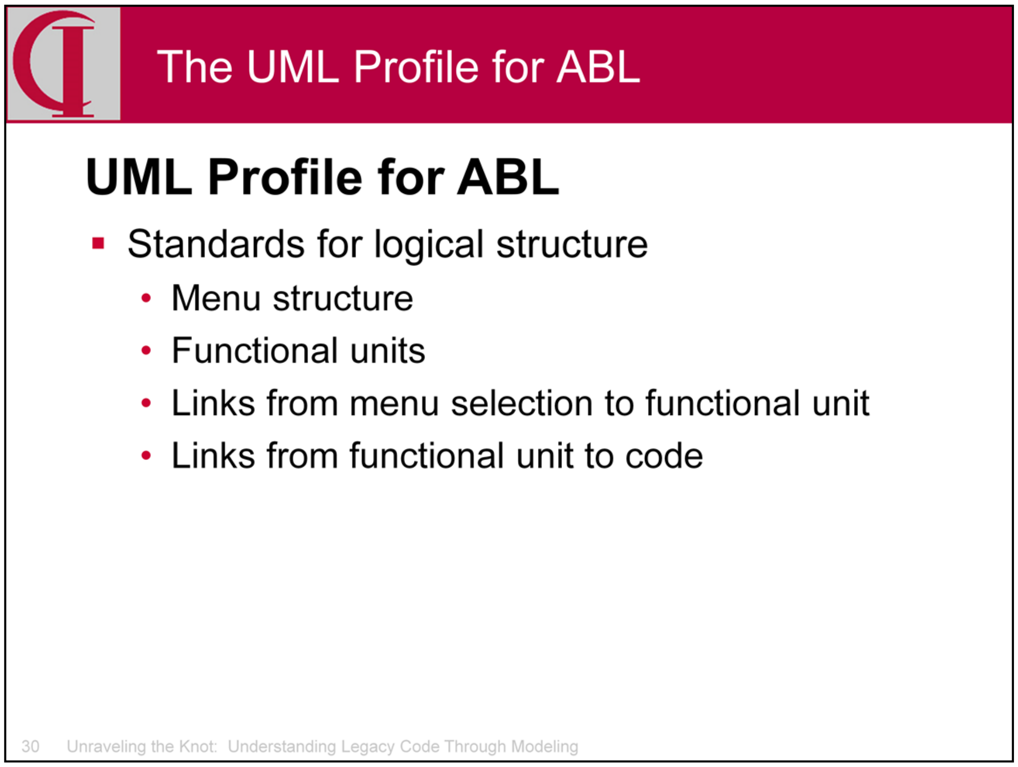
«oeProgram»
«oeIPPrivate»
prolint/prolintdb/findsessions.w: OpenQ
prolint/prolintdb/findsessions.w
«oeIPPrivate»
prolint/prolintdb/findsessions.w: enable_UI
«oeIPPrivate»
prolint/prolintdb/findsessions.w: PublishResults
«oeIPPrivate»
prolint/prolintdb/findsessions.w: SetSession
«oeTable»
lint_session
«oeProgram»
prolint/prolintdb/lintstats.p
«oeIPPrivate»
prolint/prolintdb/lintstats.p: openqSessions
«oeProgram»
«oeIPPrivate»
prolint/outputhandlers/prolintdb.p: Prolint_Status_FileStart
«oeIPPrivate»
prolint/outputhandlers/prolintdb.p: Prolint_FinalizeResults

Demo

29 Unraveling the Knot: Understanding Legacy Code Through Modeling

Here is a simple UML diagram showing one of the database tables for ProLint in pink in the center with the detail data access from internal procedures and main program bodies. The two solid lines in the lower right indicate modify access from those two internal procedures. All other links are read only and are shown as dashed lines, although that may not be clear in this reproduction. With diagrams like this, it is very easy and fast to see what is going on in the relationship of the tables and programs.


«oeProgram»
«oeIPPrivate»
prolint/prolintdb/findsessions.w: OpenQ
prolint/prolintdb/findsessions.w
«oeIPPrivate»
prolint/prolintdb/findsessions.w: enable_UI
«oeIPPrivate»
prolint/prolintdb/findsessions.w: PublishResults
«oeIPPrivate»
prolint/prolintdb/findsessions.w: SetSession
«oeTable»
lint_session
«oeProgram»
prolint/prolintdb/lintstats.p
«oeIPPrivate»
prolint/prolintdb/lintstats.p: openqSessions
«oeProgram»
«oeIPPrivate»
prolint/outputhandlers/prolintdb.p: Prolint_FinalizeResults
prolint/outputhandlers/prolintdb.p
«oeIPPrivate»
prolint/outputhandlers/prolintdb.p: Prolint_Status_FileStart

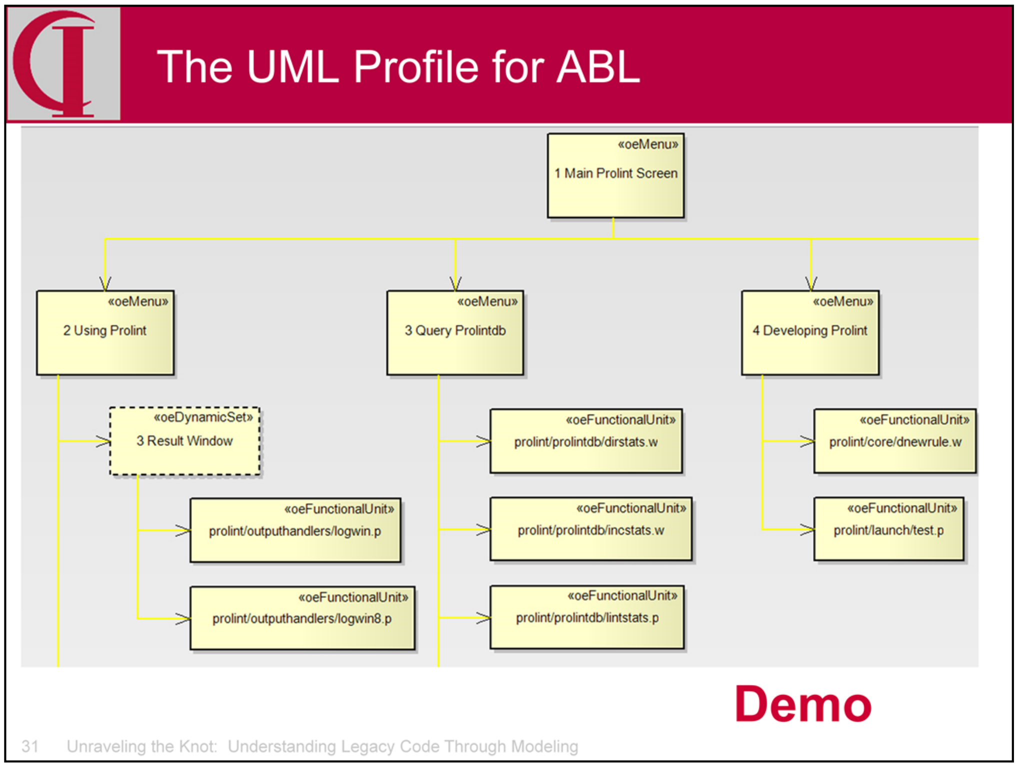

The UML Profile for ABL

UML Profile for ABL

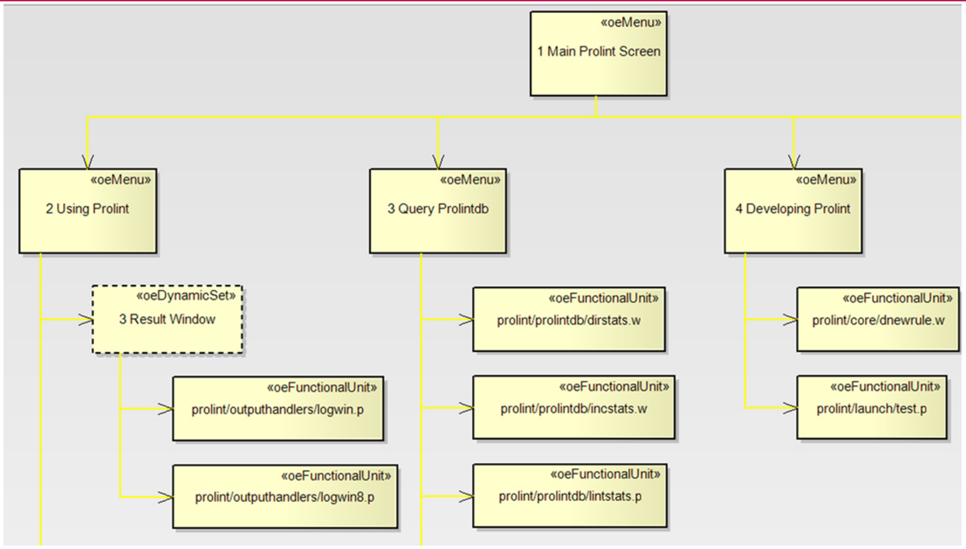
- Standards for logical structure
  - Menu structure
  - Functional units
  - Links from menu selection to functional unit
  - Links from functional unit to code

30 Unraveling the Knot: Understanding Legacy Code Through Modeling

While implemented differently in different applications, a general set of stereotypes for modeling menus is included in the Profile since they provide an important functional structure to the way in which an application is used. Each menu selection that causes the execution of something other than a menu program is linked to a Functional Unit. The Functional Unit serves as a container for all of the code which might be reached during the execution of that function. The Functional Unit is in turn linked to the Compile Unit which is the program first executed by that Functional Unit.



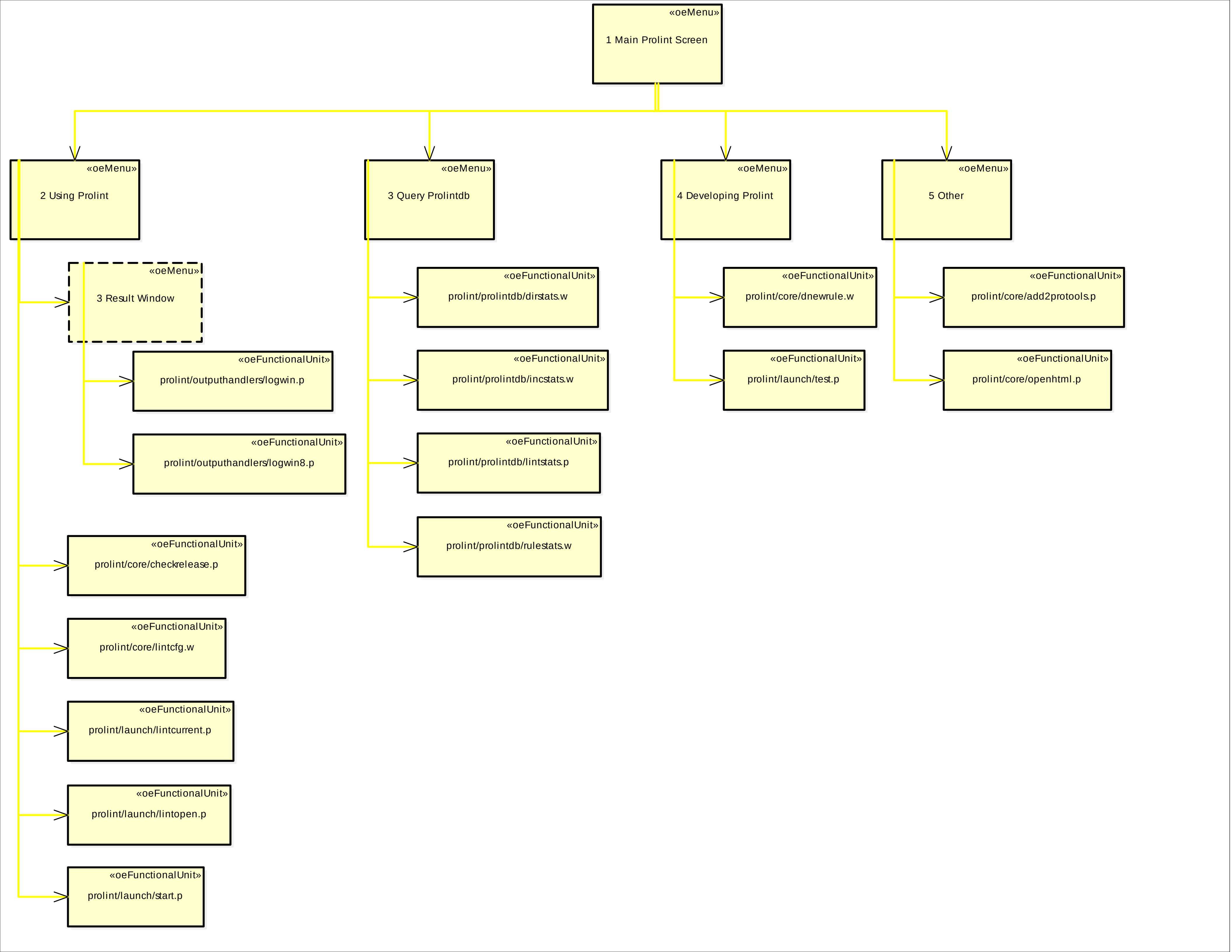
The UML Profile for ABL

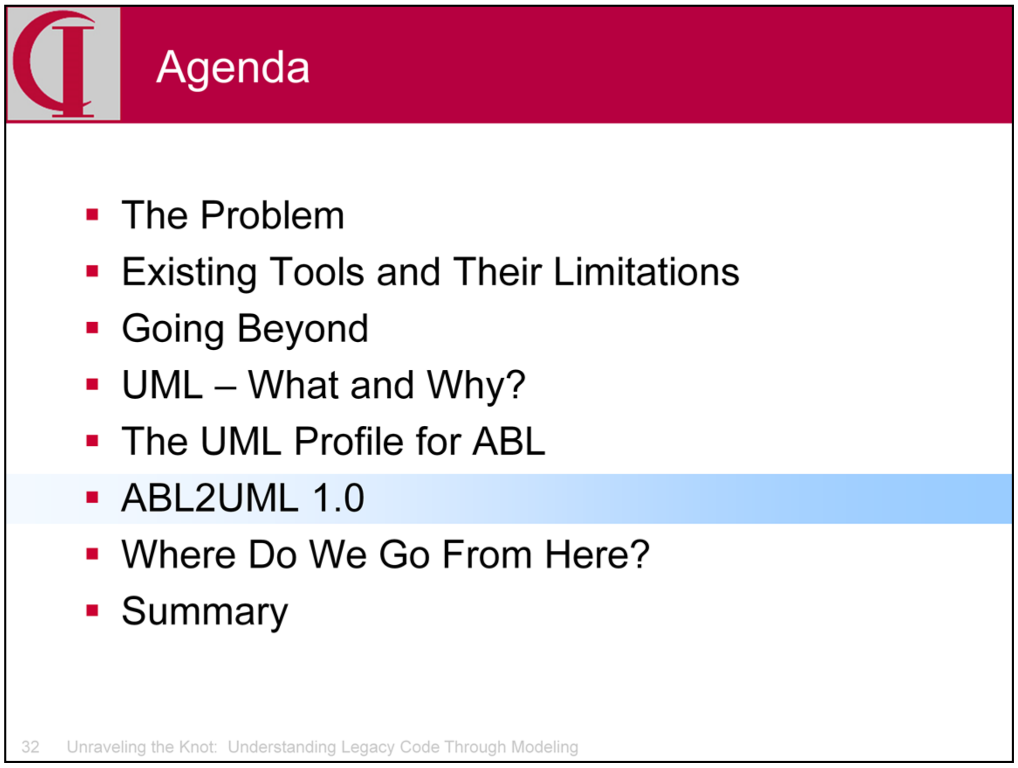


Demo

31 Unraveling the Knot: Understanding Legacy Code Through Modeling

Here we see a fragment of the menu system of ProLint with the menu selections leading to Functional Units. On the left in the dashed box is a special stereotype called a Dynamic Set for menu-like sets which are selected as a part of configuration rather than step-wise user interaction.

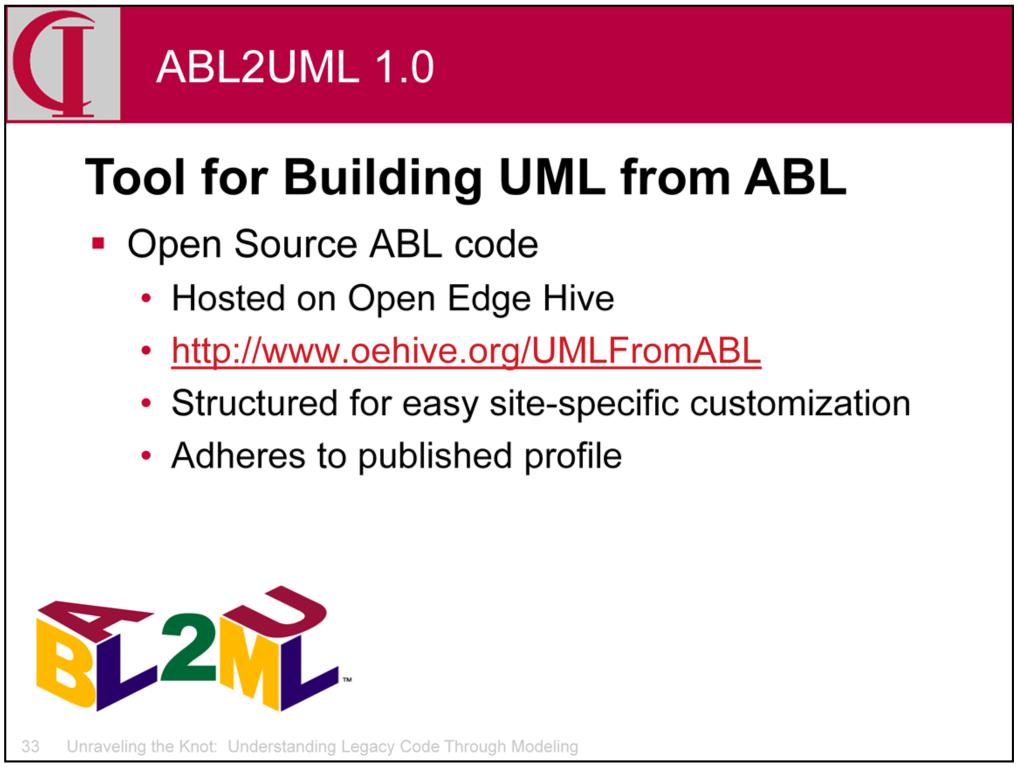

«oeMenu»
1 Main Prolint Screen
«oeMenu»
2 Using Prolint
«oeMenu»
3 Query Prolintdb
«oeMenu»
4 Developing Prolint
«oeMenu»
5 Other
«oeMenu»
3 Result Window
«oeFunctionalUnit»
prolint/outputhandlers/logwin.p
«oeFunctionalUnit»
prolint/outputhandlers/logwin8.p
«oeFunctionalUnit»
prolint/core/checkrelease.p
«oeFunctionalUnit»
prolint/core/lintcfg.w
«oeFunctionalUnit»
prolint/launch/lintcurrent.p
«oeFunctionalUnit»
prolint/launch/lintopen.p
«oeFunctionalUnit»
prolint/launch/start.p
«oeFunctionalUnit»
prolint/prolintdb/dirstats.w
«oeFunctionalUnit»
prolint/prolintdb/incstats.w
«oeFunctionalUnit»
prolint/prolintdb/lintstats.p
«oeFunctionalUnit»
prolint/prolintdb/rulestats.w
«oeFunctionalUnit»
prolint/core/dnewrule.w
«oeFunctionalUnit»
prolint/launch/test.p
«oeFunctionalUnit»
prolint/core/add2protools.p
«oeFunctionalUnit»
prolint/core/openhtml.p



Agenda

- The Problem
- Existing Tools and Their Limitations
- Going Beyond
- UML – What and Why?
- The UML Profile for ABL
- ABL2UML 1.0
- Where Do We Go From Here?
- Summary

So, having defined how we want to represent the ABL code in UML, how do we build the model?



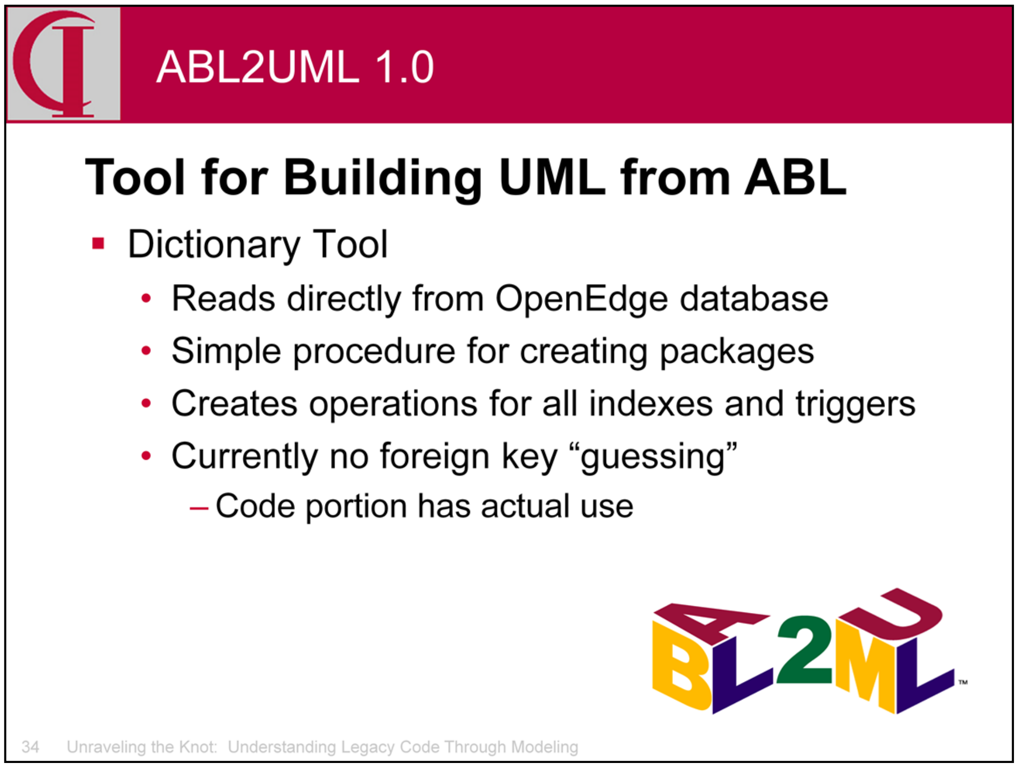
ABL2UML 1.0

Tool for Building UML from ABL

- Open Source ABL code
  - Hosted on Open Edge Hive
  - http://www.oehive.org/UMLFromABL
  - Structured for easy site-specific customization
  - Adheres to published profile

33 Unraveling the Knot: Understanding Legacy Code Through Modeling

Based on this profile, I have written ABL code which reads from a variety of sources and builds a UML model by directly writing into an OpenEdge repository for Enterprise Architect. This is an open source project based on the principle that sharing tools and adhering to common standards such as the published profile will get us all a lot farther than if each of us just builds our own tools. I have taken special care in the writing to make the code clear and easy to modify for the specifics of an individual site and will be publishing multiple versions of the site-specific code to illustrate how different schema and menu structures can be adapted to the common stereotypes. Most of the code -- all the really hard parts -- should be standard across all sites. This version is mostly not OO.



ABL2UML 1.0

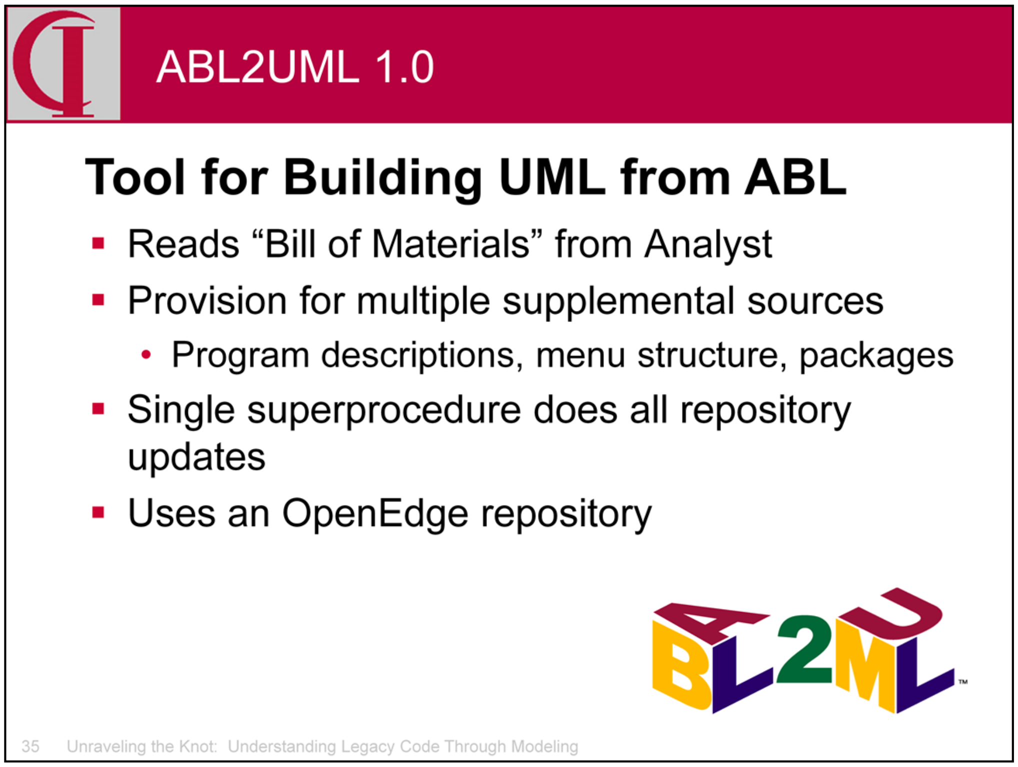
Tool for Building UML from ABL

- Dictionary Tool
  - Reads directly from OpenEdge database
  - Simple procedure for creating packages
  - Creates operations for all indexes and triggers
  - Currently no foreign key “guessing”
    - Code portion has actual use

34 Unraveling the Knot: Understanding Legacy Code Through Modeling

The schema portion of the tool reads directly from an OpenEdge database to build the model. If one has a large number of tables and a mechanism by which to group them into packages, there is a piece of the custom site code for this purpose. All details about tables, all columns, all details about columns, all indices, the primary key, and all triggers are captured. The code doesn’t currently handle data server aspects, but this is an easy adaptation with pre-existing stereotypes, if there is demand.

The related tool on PSDN has a component which will guess at foreign key relationships between tables. This capability is currently omitted from the current code on OE Hive since we are capturing actual use and WHERE clauses from the code portion. Again, if there is demand, I will add a foreign key guessing tool, written in ABL so that it is easily customized, and with structures to make it easy to omit selected columns from comparisons, e.g., audit trail fields, and to manipulate column names prior to matching, e.g., removing table prefixes.



ABL2UML 1.0

Tool for Building UML from ABL

- Reads “Bill of Materials” from Analyst
- Provision for multiple supplemental sources
  - Program descriptions, menu structure, packages
- Single superprocedure does all repository updates
- Uses an OpenEdge repository

35 Unraveling the Knot: Understanding Legacy Code Through Modeling

The code portion is primarily based on the “bill of materials” XML file produced by Analyst, but the site-specific portion is easily tailored to fit local menu systems, list of program descriptions, standard naming conventions, or whatever might be available to assist in making the model more meaningful. There is also the option of using additional custom stereotypes and custom post-processing where needed.

The code stage consists of basically five phases:

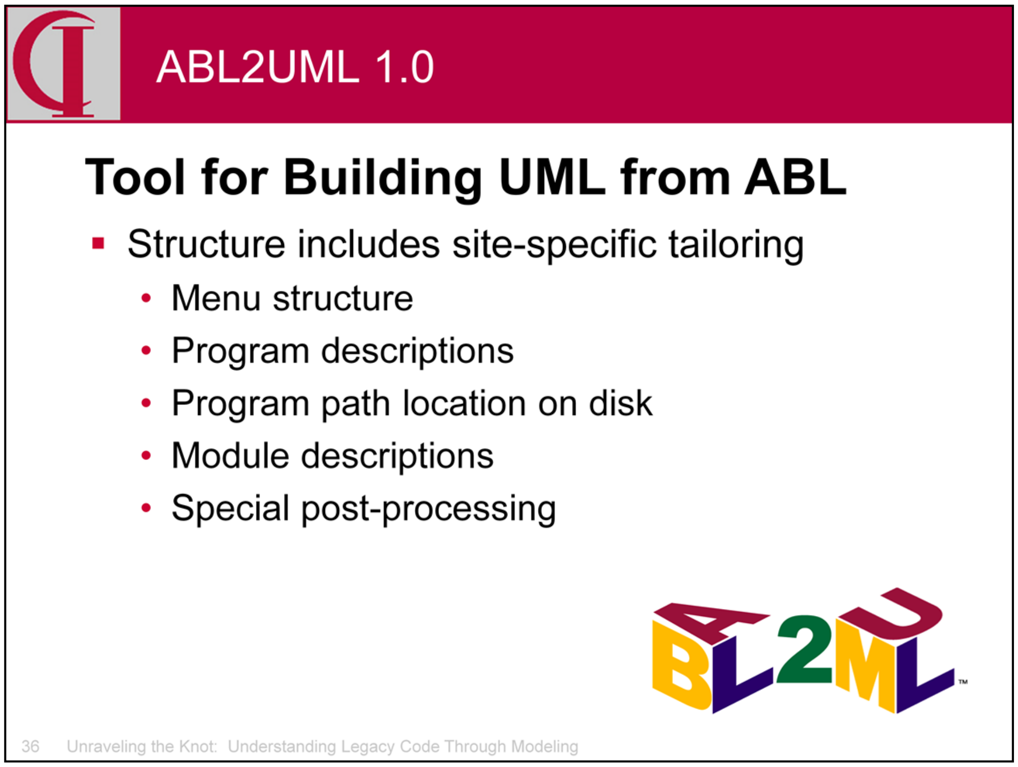
1. Building components from the bill of materials;

2. Building connectors between the components from the bill of materials;

3. Postprocessing to identify private versus public access;

4. Optional custom additions; and

5. Optional custom postprocessing for analysis similar to private versus public.

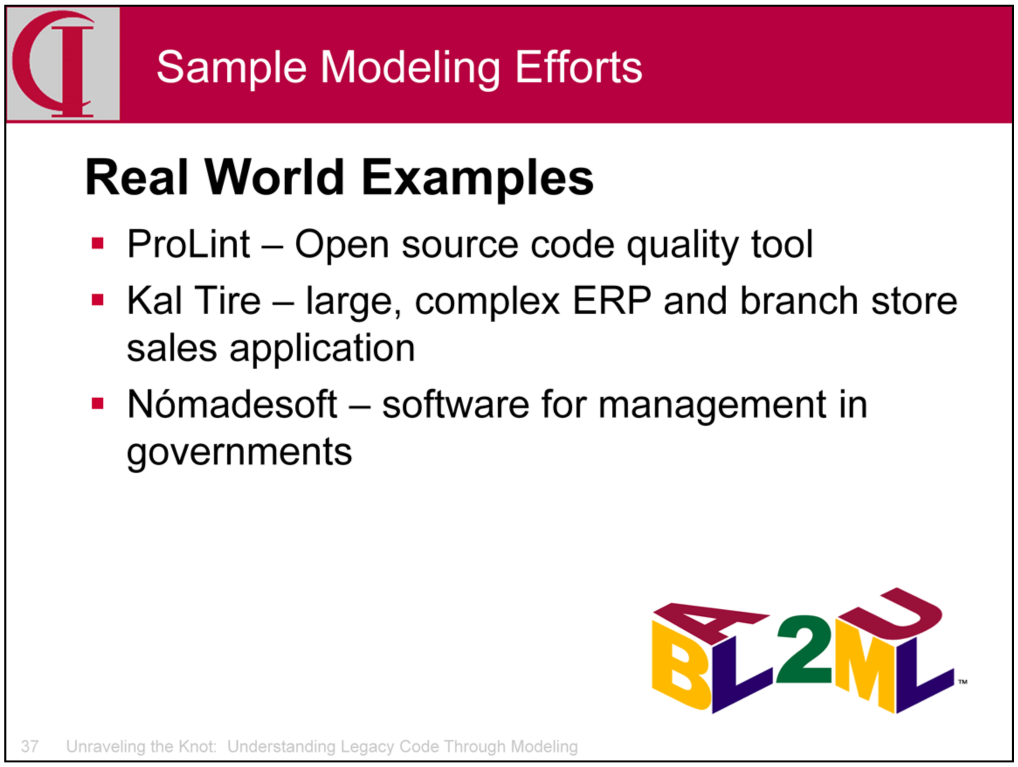


ABL2UML 1.0

Tool for Building UML from ABL

- Structure includes site-specific tailoring
  - Menu structure
  - Program descriptions
  - Program path location on disk
  - Module descriptions
  - Special post-processing

As noted, there is a separate program for custom processing with a skeleton structure and multiple examples of how to adapt the usage to site-specific implementations.



Sample Modeling Efforts

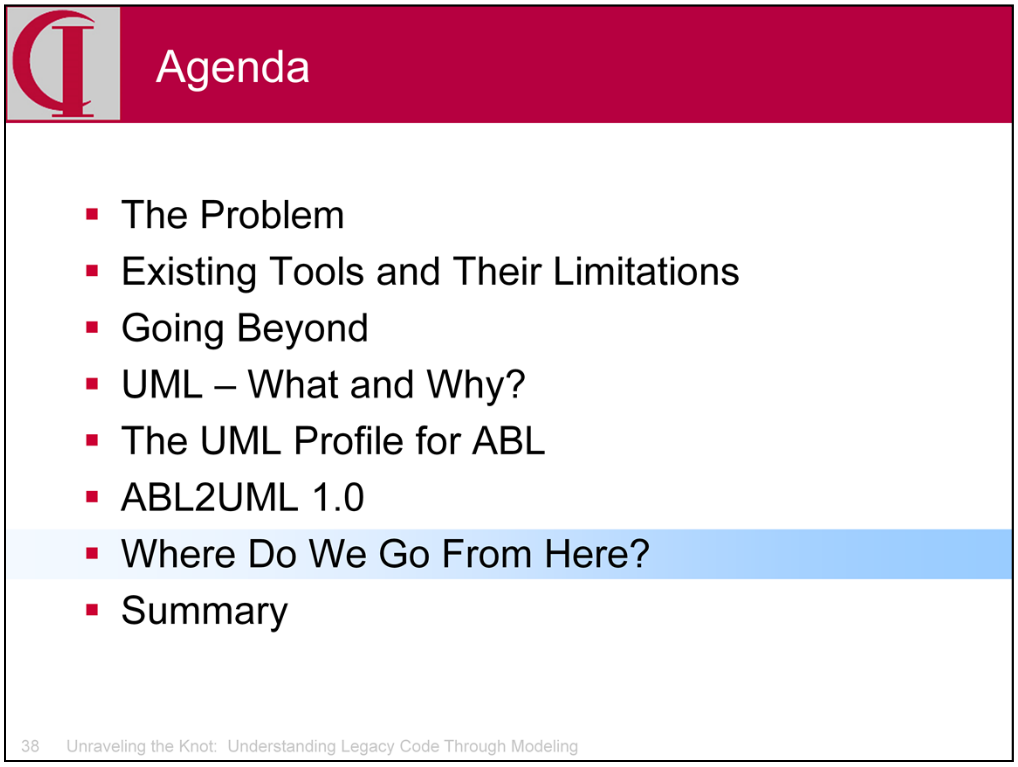
Real World Examples

- ProLint – Open source code quality tool
- Kal Tire – large, complex ERP and branch store sales application
- Nómadesoft – software for management in governments

37 Unraveling the Knot: Understanding Legacy Code Through Modeling

As you have seen, we have applied these tools to ProLint, a small but quite complex tool whose source code is freely available for download.

They have also been applied to a large, complex ERP application of nearly 2 million lines of ABL running at Kal Tire in Canada and are being used by them to analyze subsystems and potential services. Nómadesoft uses it to produce documentation that they supply with their software.

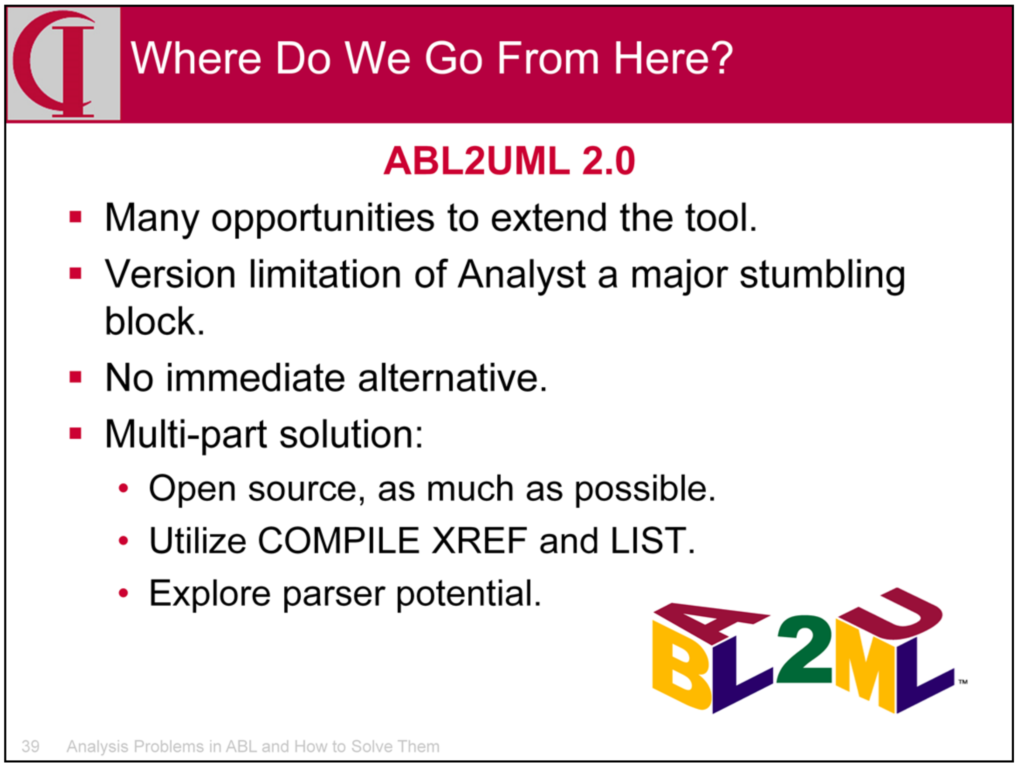


Agenda

- The Problem
- Existing Tools and Their Limitations
- Going Beyond
- UML – What and Why?
- The UML Profile for ABL
- ABL2UML 1.0
- Where Do We Go From Here?
- Summary

38 Unraveling the Knot: Understanding Legacy Code Through Modeling

So far I have been talking about the original release of the tool. Now let's consider where to go from here.



Where Do We Go From Here?

ABL2UML 2.0

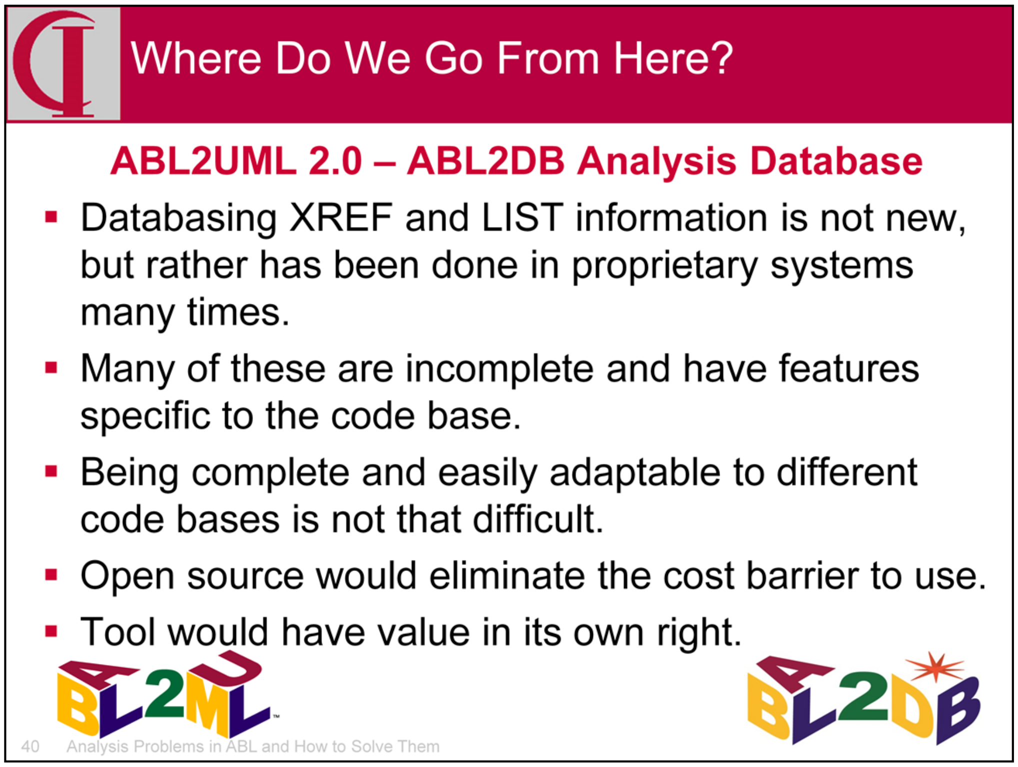
- Many opportunities to extend the tool.
- Version limitation of Analyst a major stumbling block.
- No immediate alternative.
- Multi-part solution:
  - Open source, as much as possible.
  - Utilize COMPILE XREF and LIST.
  - Explore parser potential.

ABL2UML™

39 Analysis Problems in ABL and How to Solve Them

The previously released version of this tool is an interesting and valuable start, but also clearly just the beginning. I have had many ideas for adding to its capabilities. But, I have been very concerned about the dependence on Analyst. It is a wonderful tool, but it failed to thrive in the marketplace and the discontinuation of maintenance for new versions has provoked an increasing crisis, particularly with the accelerating delivery of new versions from PSC. While many sites may be primarily version 9 or even earlier in the code base, there is a real problem if the tool is going to fall down the minute one uses a single new verb. Two things have come together this year to change that.

One is that I had thought for some time to create an open source tool which would database XREF and LIST data as completely as possible. This clearly would cover a lot of the need in the current tool, albeit with some notable exceptions. In fact, the version used by Nomadesoft is based on an XREF database rather than Analyst which was adequate for their needs. The other development this year was the work by Gilles Querret with the parser from Progress Development Studio. Let's explore these a bit further.

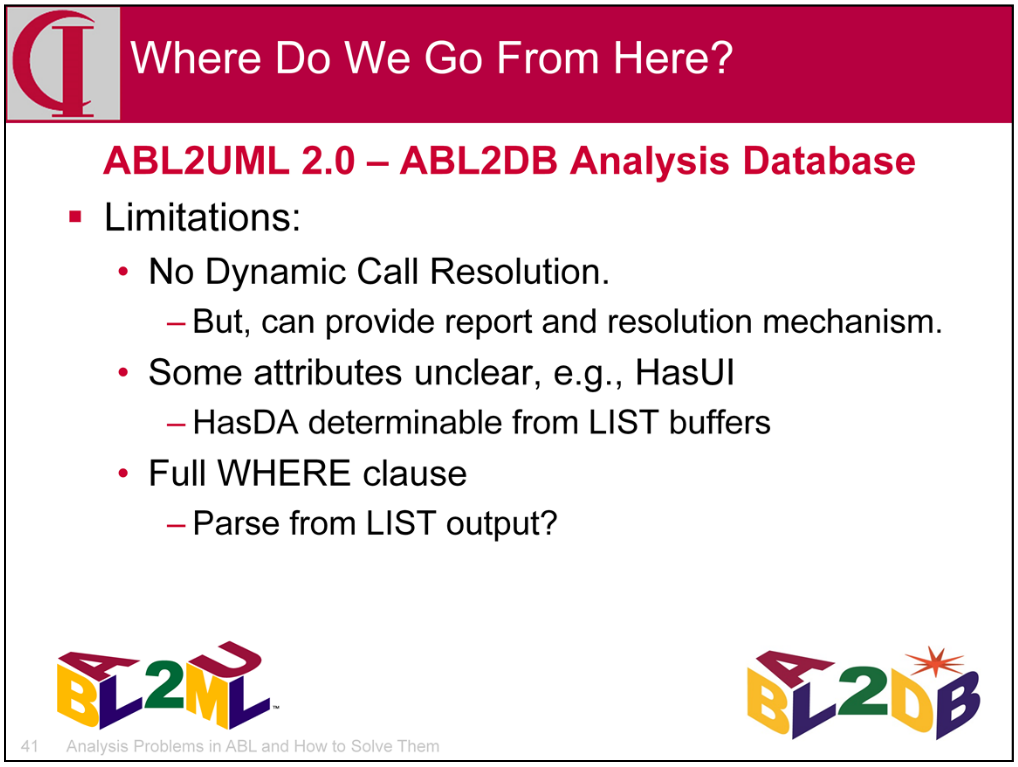


Where Do We Go From Here?

ABL2UML 2.0 – ABL2DB Analysis Database

- Databasing XREF and LIST information is not new, but rather has been done in proprietary systems many times.
- Many of these are incomplete and have features specific to the code base.
- Being complete and easily adaptable to different code bases is not that difficult.
- Open source would eliminate the cost barrier to use.
- Tool would have value in its own right.

Creating a database of XREF data is hardly a new idea. Indeed, it has been done many times by various ISVs for their own codebase. But, these are not generally available, often incomplete for what they might cover, and have features specific to the code base of that ISV, e.g., picking up the first line of the code as the description or how to read menu information. Nevertheless, making a version which is complete and which is structured for easy configuration to the specific code base does not seem that difficult. I have had some experience with the ABL2UML code itself doing this kind of thing. And, making it open source might make it easier for people to start using it, particularly since the tool would have value in its own right.



Where Do We Go From Here?

ABL2UML 2.0 – ABL2DB Analysis Database

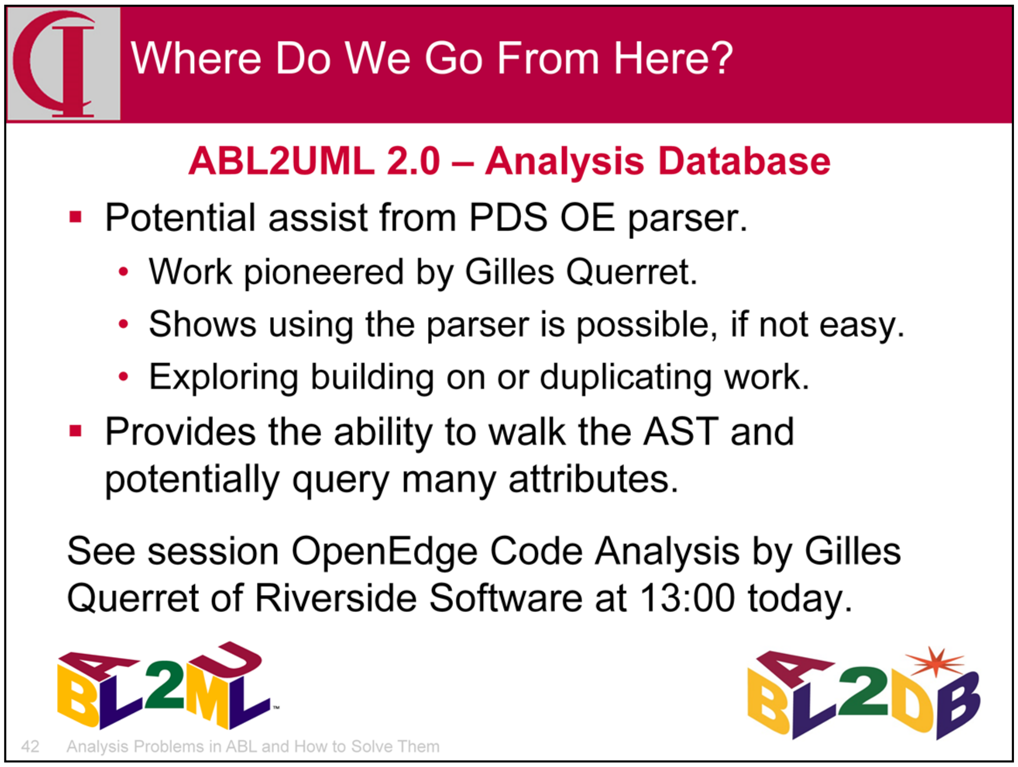
- Limitations:
  - No Dynamic Call Resolution.
    - But, can provide report and resolution mechanism.
  - Some attributes unclear, e.g., HasUI
    - HasDA determinable from LIST buffers
  - Full WHERE clause
    - Parse from LIST output?

41 Analysis Problems in ABL and How to Solve Them

Relying solely on XREF and LIST data is going to have some limitations. Most dramatic among these will be the lack of dynamic call resolution since this is dependent on call graph analysis, which is beyond what we can expect to include in an initial release at least. However, one can certainly provide an unresolved dynamic call report and a mechanism for manual resolution. Depending on the code base, this will vary from easy to difficult to complete.

There are also some attributes in the current profile which may be less easy to identify. One of these is the HasUI property. The HasDA property we can probably provide by parsing the COMPILE LIST output for buffers.

Also, the current profile includes the full WHERE clause for an data access. It may be possible to parse this from the COMPILE LIST output using indicators from the COMPILE XREF output, but this may not be easy.



Where Do We Go From Here?

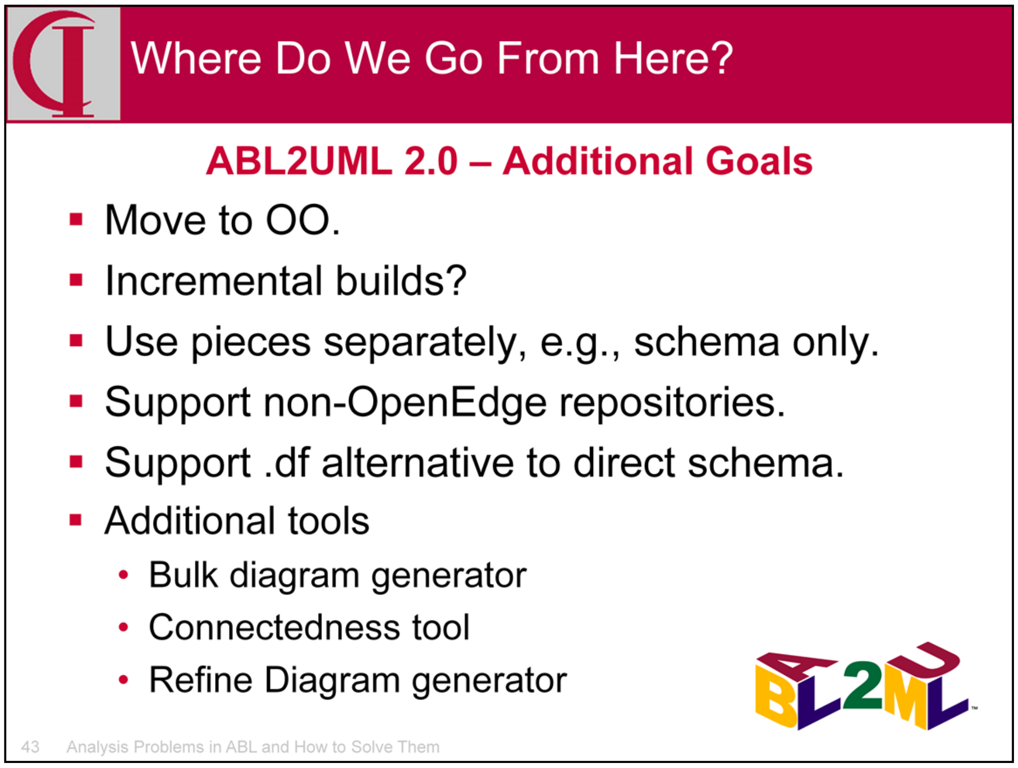
ABL2UML 2.0 – Analysis Database

- Potential assist from PDS OE parser.
  - Work pioneered by Gilles Querret.
  - Shows using the parser is possible, if not easy.
  - Exploring building on or duplicating work.
- Provides the ability to walk the AST and potentially query many attributes.

See session OpenEdge Code Analysis by Gilles Querret of Riverside Software at 13:00 today.

42 Analysis Problems in ABL and How to Solve Them

To supplement what one can get from the COMPILE options, I am going to explore what can be done with the parser in Progress Development Studio. Gilles Querret has shown that it is possible to use this to access the Abstract Symbol Tree in much the same way that PDS does to provide syntax aware coloration and the like. He is using it to provide input to Sonar Qube, an open source code analysis tool. This should allow fairly quickly removing all limitations except the dynamic call resolution and provides the foundation for many possible additions.



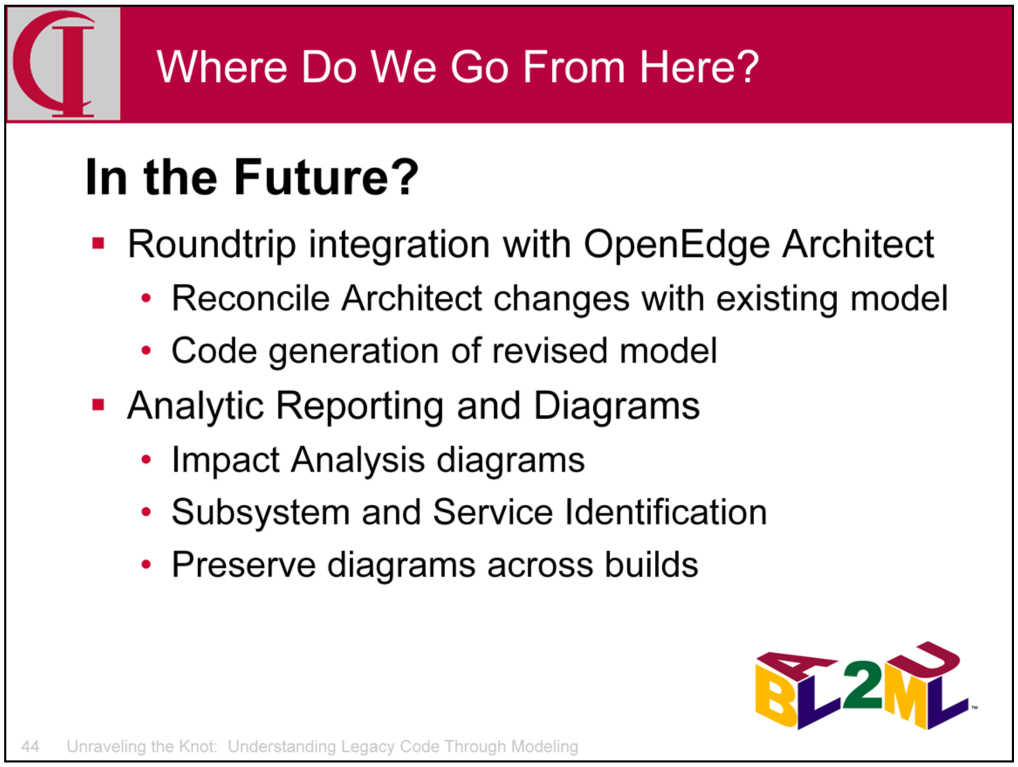
Where Do We Go From Here?

ABL2UML 2.0 – Additional Goals

- Move to OO.
- Incremental builds?
- Use pieces separately, e.g., schema only.
- Support non-OpenEdge repositories.
- Support .df alternative to direct schema.
- Additional tools
  - Bulk diagram generator
  - Connectedness tool
  - Refine Diagram generator

43 Analysis Problems in ABL and How to Solve Them

In addition to the move to an open source database and possible parser, there are some addition goals in the current work. I want to move the tool to OO because it is the right thing to do. When first written, 10.1A was new and I was concerned about how many people would have that recent a version. Now, I tend to think that one's development and analysis platform should be current, even if the production code isn't. I would like to provide for incremental builds, although full builds provide a certain confidence. I want to make it easier and more obvious how to use pieces separately, e.g., just the schema. While there is appeal about building directly to an OpenEdge database, I want to allow building to other technology respository databases and even non-EA products. The current product builds directly from the schema, but I would like to add building from a .df including incremental. And, I want to consider further tools such as a bulk generator for diagrams of specified types, a connectedness assessment tool, and to finalize the existing diagram generator.



Where Do We Go From Here?

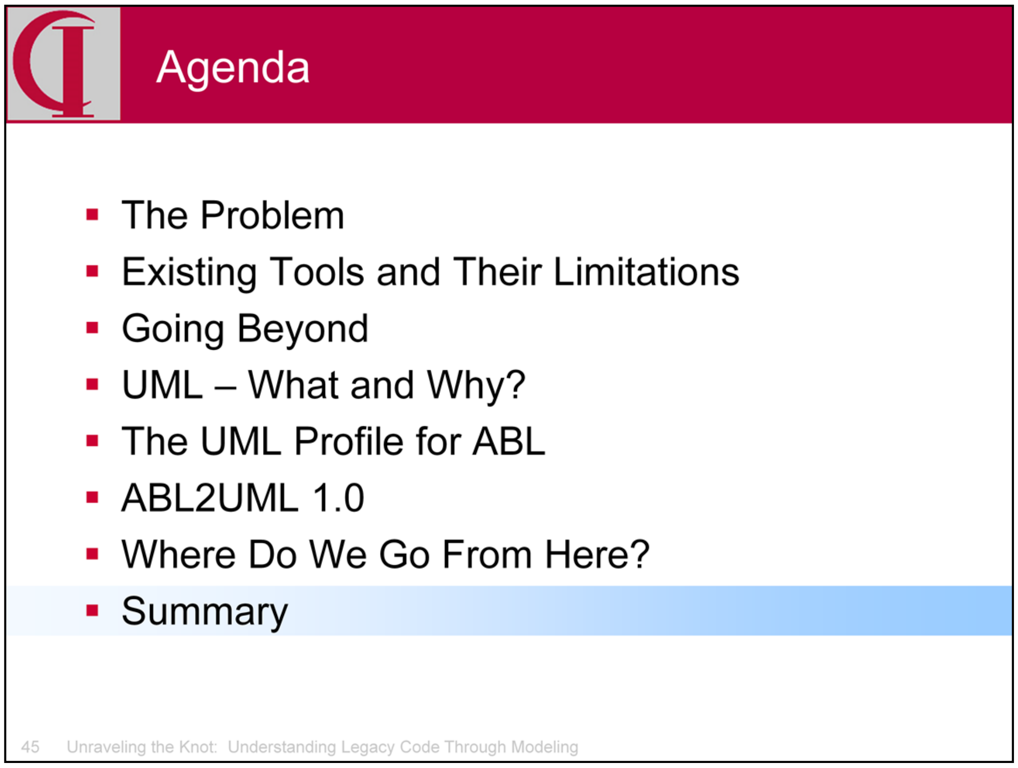
In the Future?

- Roundtrip integration with OpenEdge Architect
  - Reconcile Architect changes with existing model
  - Code generation of revised model
- Analytic Reporting and Diagrams
  - Impact Analysis diagrams
  - Subsystem and Service Identification
  - Preserve diagrams across builds

44 Unraveling the Knot: Understanding Legacy Code Through Modeling

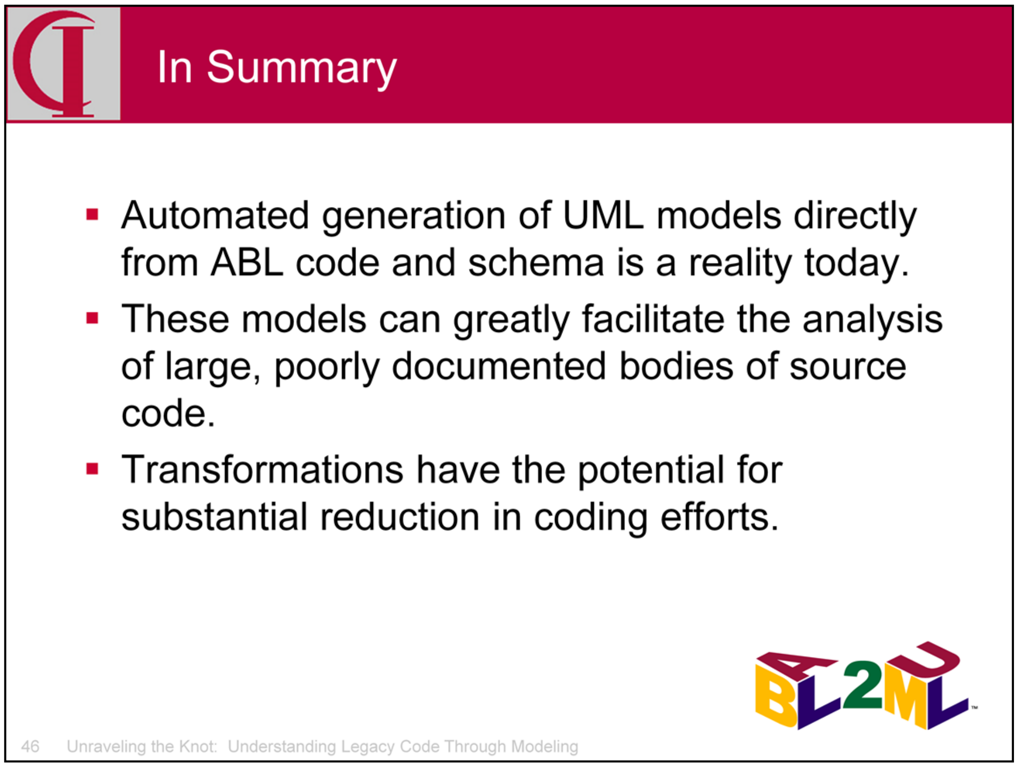
I am also looking at code generation from a revised model. Initially, this will not be anything like full transformation, but simply being able to make small changes in the model and then produce revised code. This will be connected to roundtrip integration with OpenEdge Architect, not just for OO code, but any code in a legacy system. While a primary target will be OpenEdge Architect, I will be looking at ways to make this available to those using other development tools as well.

I will also be working on various forms of impact analysis and subsystem and service identification tools.



Agenda

- The Problem
- Existing Tools and Their Limitations
- Going Beyond
- UML – What and Why?
- The UML Profile for ABL
- ABL2UML 1.0
- Where Do We Go From Here?
- Summary



In Summary

- Automated generation of UML models directly from ABL code and schema is a reality today.
- These models can greatly facilitate the analysis of large, poorly documented bodies of source code.
- Transformations have the potential for substantial reduction in coding efforts.

Automated building of UML models from ABL code and schema is available today and in production at a large site. This is a more comprehensive and complete tool than any prior efforts of which I am aware and is a tool that is likely on its own to provide substantial value to anyone trying to come to grips with a large legacy code base. This information will also provide important information for any transformation project, shortening analysis time and reducing costs. I invite anyone with these needs to contact me so that I can help them figure out how to adapt the tools to their specific code and how to move forward with their projects.

While this is itself an important milestone, it is only the beginning and there will be many additional capabilities made available in the coming months. Stay tuned at the OpenEdge Hive for developments or contact me for anything which is of particular interest.

For More Information, go to…

- OpenEdge Hive
  - This project: http://www.oehive.org/UMLFromABL
  - Enterprise Architect with OpenEdge http://www.oehive.org/EA
  - ProLint http://www.oehive.org/prolint
  - ProRefactor http://www.oehive.org/prorefactor
- Joanju Analyst
  - http://www.joanju.com/analyst/
- Sparxsystems
  - Enterprise Architect http://www.sparxsystems.com/products/index.html
  - UML Tutorial http://www.sparxsystems.com/resources/uml2_tutorial/

Questions?

Thank You

Dr. Thomas Mercer-Hursh
thomas@cintegrity.com