


DEV-35: Modeling Existing ABL Systems with UML

Thomas Mercer-Hursh, Ph.D.
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 a major step forward in our ability 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
- The UML Toolset for ABL
- Sample Modeling Efforts
- Where Do We Go From Here?





The Problem

- Legacy ABL System
- Millions of Lines of ABL Code
- Years and Years of Modifications
- Documentation?



Many ABL systems are 10 or even 20 years old.

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 is just 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 happily for many years enjoying the low cost of ownership.



The Problem

And then **Crisis!**

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



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
- The UML Toolset for ABL
- Sample Modeling Efforts
- Where Do We Go From Here?





ERD Tools

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



Many shops have used ERD 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.



XREF Tools

- Information on Code, not Schema
- Index but no WHERE Clause
- “Per each” analysis only
- No Dynamic Call Resolution

Database can provide limited Where Used data



To complement ERD tools, many shops have built tools which analyze the output of COMPILE XREF, in many 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 and 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.

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 a 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
- The UML Toolset for ABL
- Sample Modeling Efforts
- Where Do We Go From Here?





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



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, but 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.



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]



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 be using that same code.

These same issues exist for function calls.



The Solution

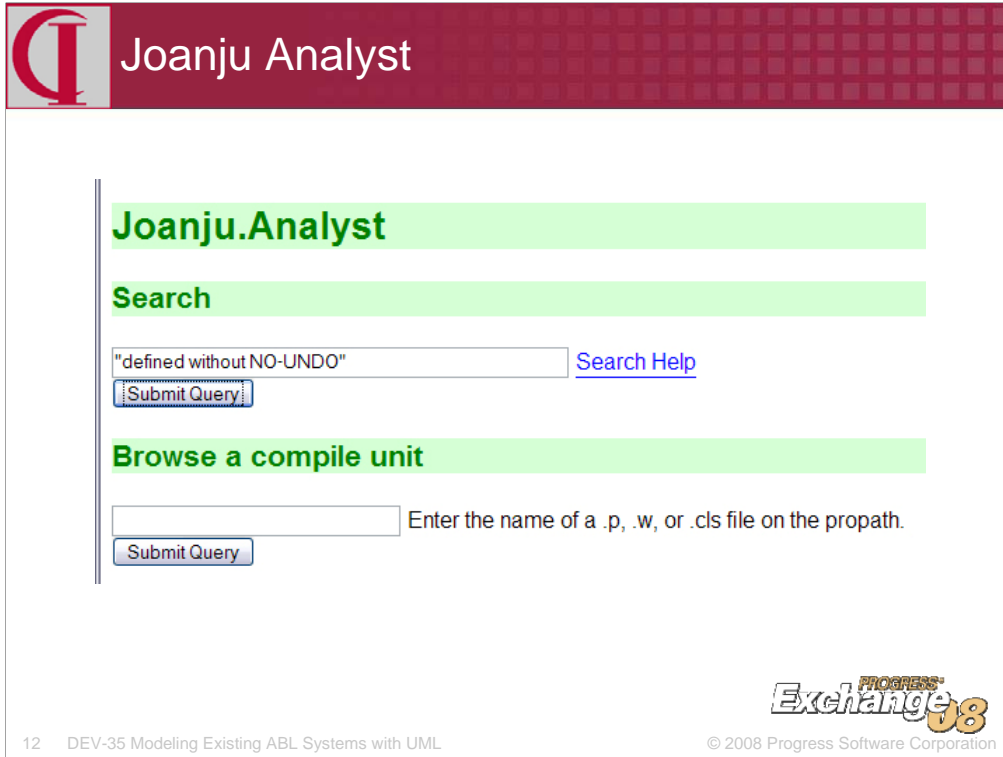
- Joanju Analyst
 - Built on Proparse and ProRefactor
 - Resolves many types of calls automatically
 - Produces an unresolved call report
 - Provides for “hints” to resolve all calls
 - Results in HTML pages with links
 - Also produces a “bill of materials” XML output



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 following 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.



The Analyst search and browse facility can search for arbitrary strings, such as shown here, or searches can be qualified to look specifically for includes, calls, tables, or fields. The browse facility is used to look at specific compile units.

The search facility will return a list of compile units containing the specified search target. Clicking on those will open up a screen on the compile unit.

Because ProLint is open source and freely available from the OpenEdge Hive (<http://www.oehive.org/prolint>) the examples which follow are based on the analysis and modeling of the source code of ProLint. Further examples will soon be published based on AutoEdge, PSDN's Reference Implementation.



Rendering [/work2/prolint72/prolint/rules/noundo.p ...](#)

[Front page](#)

Include files:

[Collapse all](#)

[Expand all](#)

```

define variable SessionWindowSystem as character no-undo.
if session: BATCH-MODE then
assign
    SessionWindowSystem = ""
.
else
assign
    SessionWindowSystem = session: DISPLAY-TYPE
.
define input parameter xreffile as character no-undo.
define input parameter listingfile as character no-undo.
define input parameter hLintSuper as handle no-undo.
define input parameter hparser as handle no-undo.
define input parameter hTopnode as integer no-undo.
```



Here you see Analyst dynamically rendering an individual compile unit.

This source is preprocessed, stripped of comments, and pretty printed.

Clicking on the link at the top, presents options to view the original source and to open any procedures which call the current one. Opening those procedures will open on the line where the call occurs. Both types of links can use the facilities of tabbed browsers for optimum navigational ease.

Note that include files have been expanded in line and can be collapsed or expanded dynamically, either individually or as a whole. One can also link to the original source file for the include.



Joanju Analyst

```
if havevar and ( not havenoundo ) then
run PublishResult ( input compilationunit, input parserGetNodeFilename (
input theNode ), input parserGetNodeLine ( input theNode ), input substitute (
"&1 '&2' defined without NO-UNDO":T, vartype, varname ), input rule_id ).
parserReleaseHandle ( input child ).
end procedure.
```

```
if havevar and ( not havenoundo ) then
run PublishResult ( input compilationunit, input parserGetNodeFilename (
 but theNode ), input substitute (
 , varname ), input rule_id ).
parserReleaseHandle ( input child ).
end procedure.
```



If one goes to the line containing our search string, which you can see complete in the top image, one can see that this string is passed as an argument to RUN PublishResult. Clicking on the run keyword, produces the drop down shown below. Note that Analyst has figured out that PublishResult is in a superprocedure and a link is provided to take one directly to the code in the superprocedure.

This is only a very brief introduction to the capabilities of Analyst and I encourage you to go try out the demo yourself at the link which will be provided at the end of the talk. Analyst on its own is a very exciting addition to our tool repertoire for ABL. But, we still don't have the standardized model we were looking for.



Agenda

- The Problem
- Existing Tools and Their Limitations
- Going Beyond
- UML – What and Why?
- The UML Profile for ABL
- The UML Toolset for ABL
- Sample Modeling Efforts
- Where Do We Go From Here?





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



UNIFIED MODELING LANGUAGE™



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 effort. 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 – 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



UML 2.0 and 2.1 have become a very rich field 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 here, 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 – Existing Tools

- Enterprise Architect
 - Widely used in Progress
 - 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



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.



Agenda

- The Problem
- Existing Tools and Their Limitations
- Going Beyond
- UML – What and Why?
- The UML Profile for ABL
- The UML Toolset for ABL
- Sample Modeling Efforts
- Where Do We Go From Here?





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.



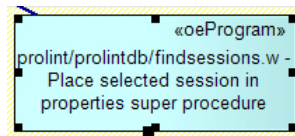
The UML Profile for ABL

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



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.



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>



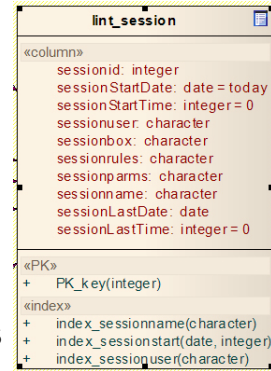
Profiles are normally defined for a particular application domain, like telephony, or for a particular form of analysis or methodology. They are not normally used for a language. However, OO languages tend to map onto UML components in a fairly simple way, but ABL, particularly legacy ABL, does not have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

Therefore, I was 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.



UML Profile for ABL

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

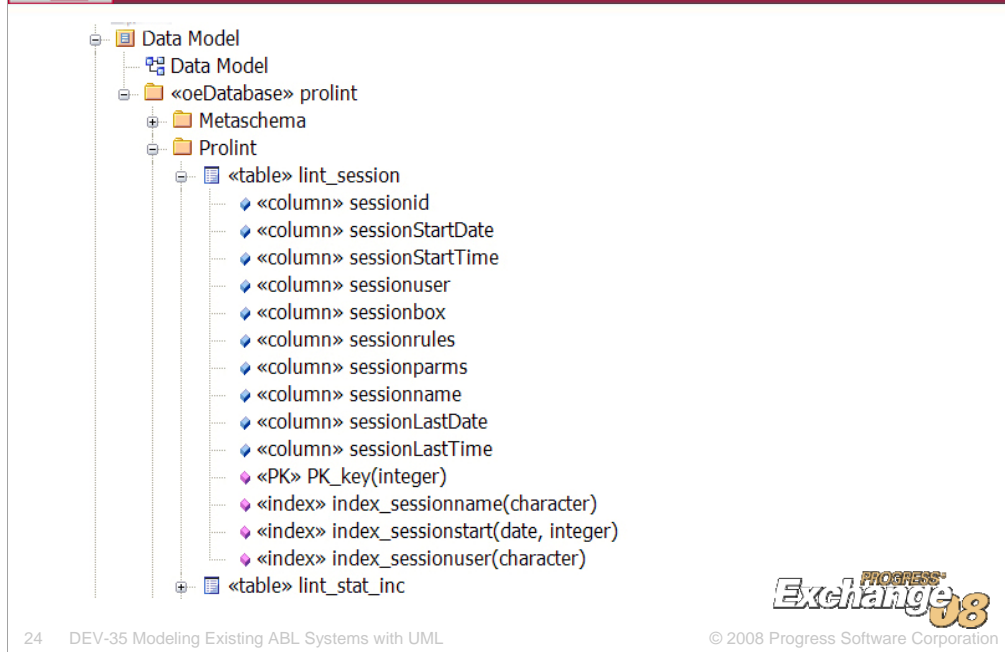


The stereotypes and tags for the data model cover all properties in the schema, including physical layout of the database, although we have not yet implemented the DataServer portions.

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 this information has been included in the profile, but I have not provided code for this in the initial ABL to UML tool, since it shows how the tables are actually used in the code.



The UML Profile for ABL

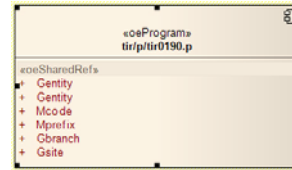


Here is a quick look at a piece of the Data Model for ProLint as it is displayed in the Project Browser panel of Enterprise Architect. As noted, all UML examples in this presentation are taken from an analysis of ProLint. Here one can see that a single database has been divided into packages for Metaschema 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 but which are available by looking at the details of the table object.



UML Profile for ABL

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



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 ...



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.

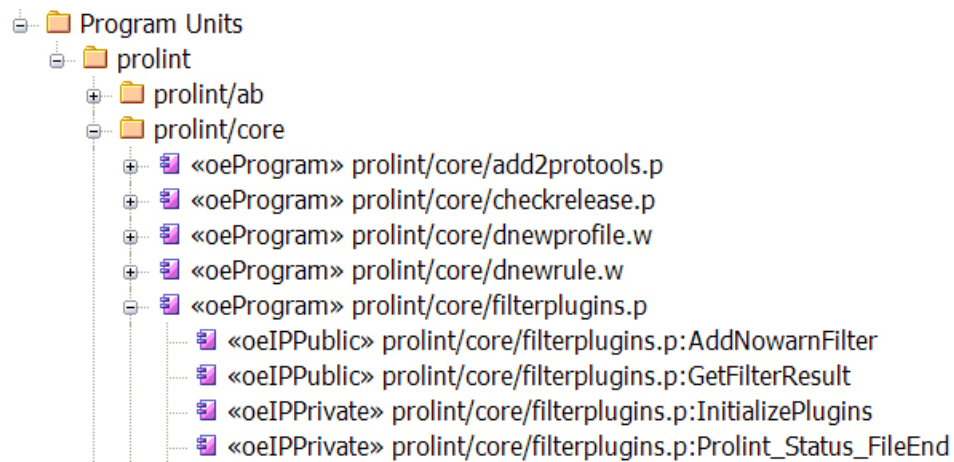
As a part of the analysis in building the model, internal procedures and functions are classified as public or private based on whether any of the references to them come from outside the compile unit.

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



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.

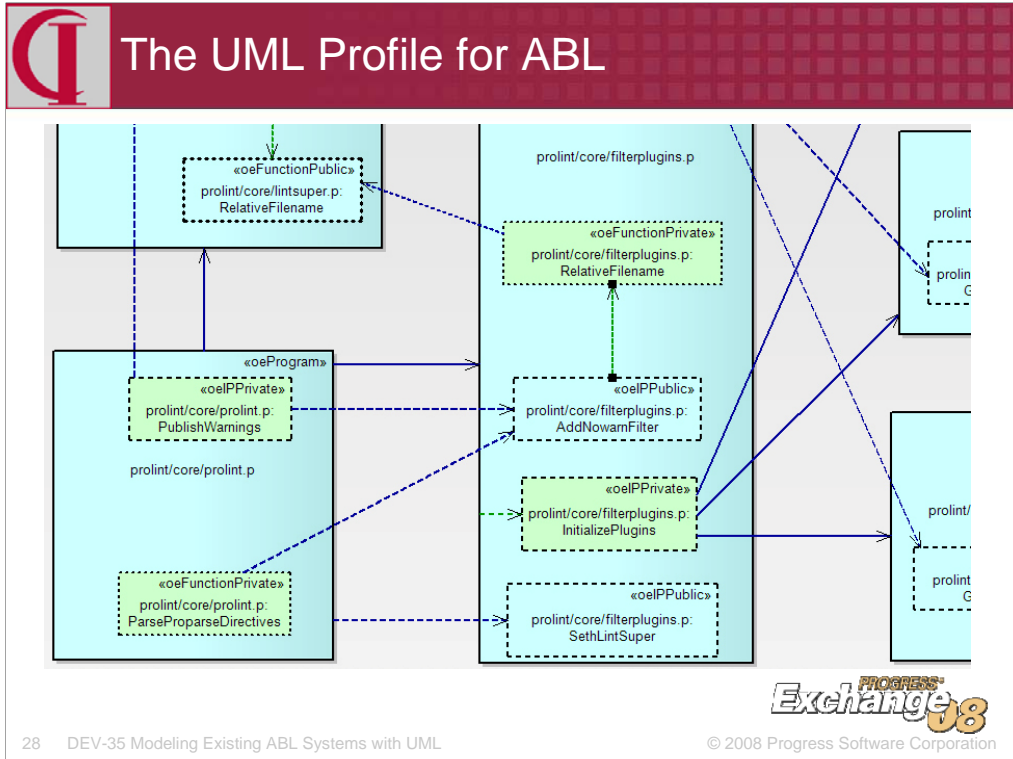


UML Profile for ABL

- 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.



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 stereotyped 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.



UML Profile for ABL

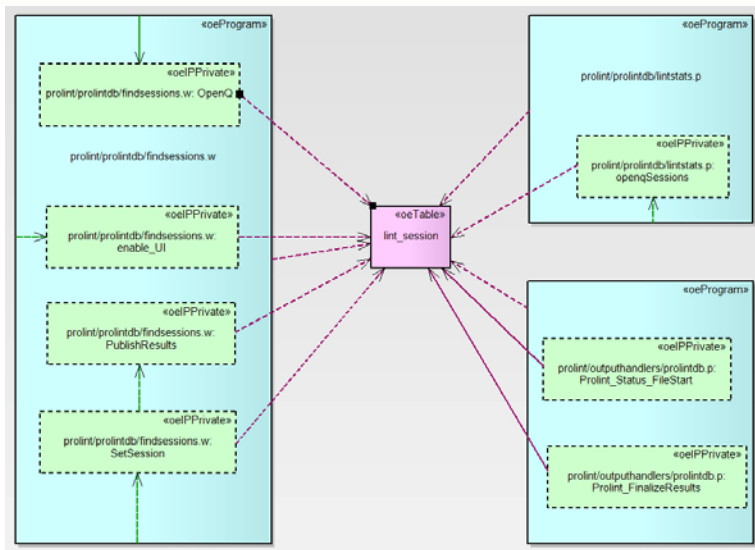
- Standards for code to data links
 - Table reference and mode
 - Column reference and mode
 - WHERE clause
 - Tied to internal procedure or function
 - Summaries by program (.p or .cls)
 - Summaries by compile unit (.r)



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



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.



UML Profile for ABL

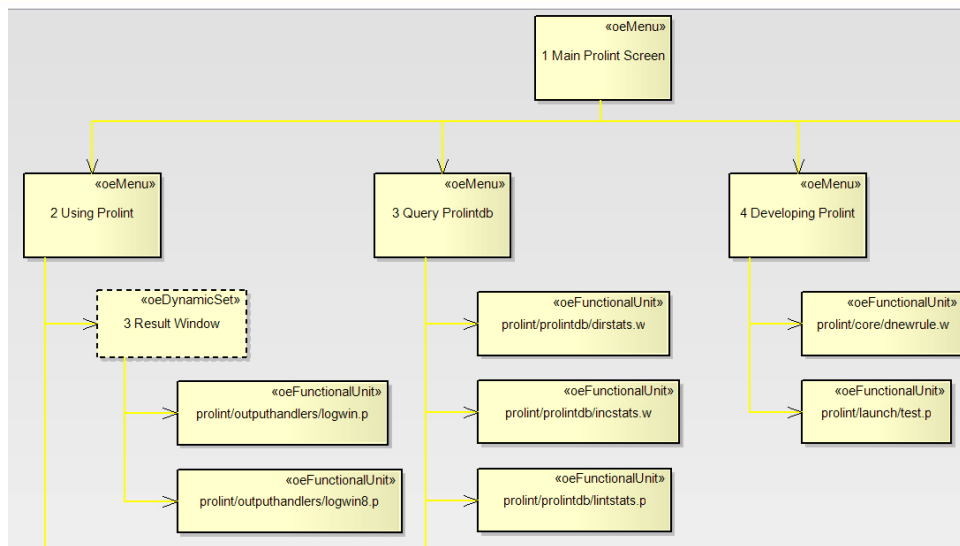
- Standards for logical structure
 - Menu structure
 - Functional units
 - Links from menu selection to functional unit
 - Links from functional unit to code



While implemented differently in different applications, a general set of stereotypes for modeling menus are 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



PROGRESS
Exchange 8

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.



Agenda

- The Problem
- Existing Tools and Their Limitations
- Going Beyond
- UML – What and Why?
- The UML Profile for ABL
- The UML Toolset for ABL
- Sample Modeling Efforts
- Where Do We Go From Here?





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



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. Currently, the code has one dependency on version 10.1A OO features, but this could be eliminated if it were necessary to work on older versions. Most of the code intentionally uses a simpler superprocedure approach in order to make it easy to understand and modify for those unfamiliar with OO.



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



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 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.



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



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.



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.



Agenda

- The Problem
- Existing Tools and Their Limitations
- Going Beyond
- UML – What and Why?
- The UML Profile for ABL
- The UML Toolset for ABL
- Sample Modeling Efforts
- Where Do We Go From Here?





Real World Examples

- Kal Tire – large, complex ERP and branch store sales application
- ProLint – Open source code quality tool
- AutoEdge – Progress' own OpenEdge Reference Architecture reference implementation
- Integrity/Solutions – large complex ERP suite focusing on distribution



These tools have 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. We have also applied it to ProLint, a much smaller but quite complex tool whose source code is freely available for download. We are well along in the process of applying it to AutoEdge, Progress' own Reference Architecture implementation, whose code is also freely available. We will soon be applying it to Integrity/Solutions, another large, complex ERP application of about 1.75 million lines of ABL and selected portions will be published on OpenEdge Hive.

The sample UML diagrams you have seen earlier were taken directly from the Enterprise Architect screen, i.e., this whole system is up and running and in production today. Additional full-sized examples will be found on OpenEdge Hive as well as sample model files.

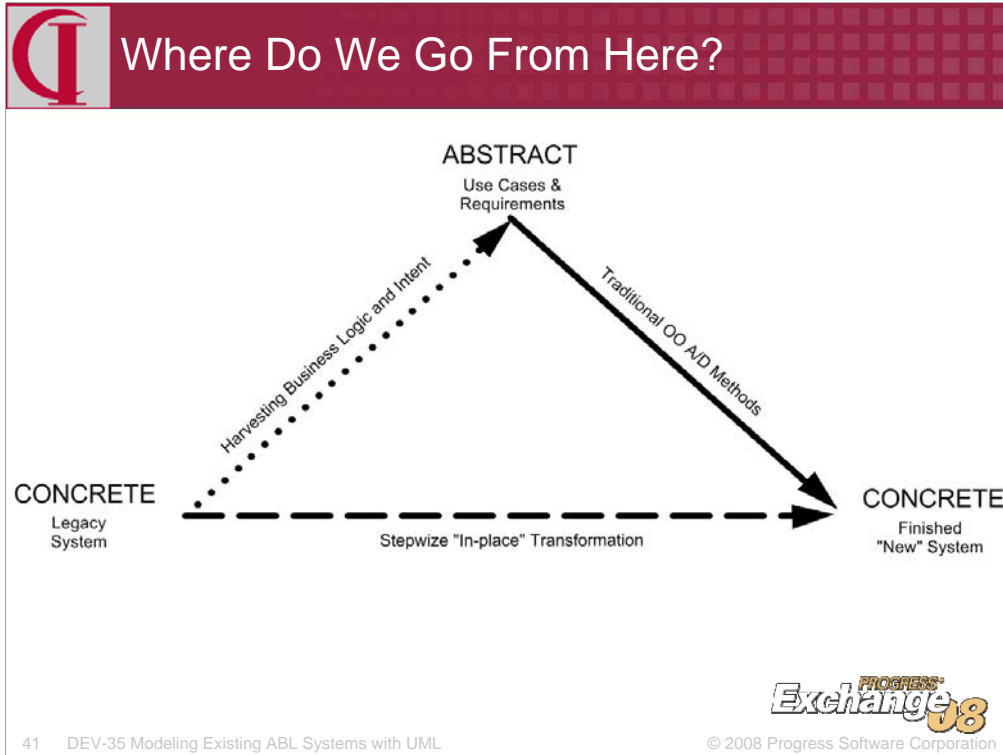
Bottom line --- here today, in use and ready for you to use on your own application.



Agenda

- The Problem
- Existing Tools and Their Limitations
- Going Beyond
- UML – What and Why?
- The UML Profile for ABL
- The UML Toolset for ABL
- Sample Modeling Efforts
- Where Do We Go From Here?





Before moving on to specifics, let me say a little about the general nature of the problem of what we are trying to do. In traditional OO Analysis and Design, we begin with diagrams and information which is very abstracted from the system we hope to build – use cases and requirements. We then refine and transform this information into progressively more concrete and specific forms until eventually we end up with the actual finished system, here in the lower right.

But, what we are doing here is starting with a concrete legacy system in the lower left. Little of the design intent or purpose is recorded here. So, we have two paths to a new system. One of these is to harvest business logic and intuit design intent in order to build the kind of abstract information which is the starting point for traditional OO A/D. This is extremely difficult and poorly understood, although if successful, will result in a new system which has the ideal architecture and form.

The alternative is to make a series of incremental, step-wise transformations of the existing system with the goal of gradually making it more and more like the system we would like to have. Such a transformation is unlikely to ever be 100% complete or to produce as “clean” a new system as one produced by the traditional OO A/D path, but it allows change to occur at a slower pace, driven by short term returns.



On-Going Open Source Project

- Localizations for additional sites
- Additional model elements as needed
- Open source Enterprise Architect plug-ins
 - Link to Analyst pages
 - Diagram generator
- Additional tools
 - Bulk diagram generator
 - Connectedness tool



With that background, let me say that this project is very much a work in progress. While much has been accomplished, there is much more coming.

I hope to create localizations for additional sites as companies become interested in the tool. As I and others develop additional tools for extracting information from the code, I will extend the profile to cover additional tags and stereotypes. One of the consultant developers at Kal Tire has been developing add-ins for Enterprise Architect which we hope to make available as they are finished. These include:

- A facility that allows a quick link from any element in the model to the HTML page for the source code related to that element;
- A tool which will follow selected link stereotypes and selected element stereotypes to automatically build a diagram relative to a given starting element;
- A tool for automatically hiding or making visible links in a diagram by stereotype; and
- An assortment of other convenient tools for selecting, deleting, and manipulating diagram elements by stereotype.

I am also working on a ABL version of the diagram generator for bulk generation of diagrams and a tool for analyzing connectedness for help in identifying subsystems and potential services in an SOA.



On-Going Open Source Project

- 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
- ProRefactor-based Super-"XREF" tool



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.

There is also a proposed project based on ProRefactor which would create a sort of super-XREF database. This might provide additional code features that could be included in the model.



On-Going Open Source Project

- Transformations via Model Driven Architecture!
 - Data Layer substitution
 - Abstraction to design model
 - UI layer separation?
 - Others ...



Of course, this is only the beginning. For some sites, the UML modeling alone will be a sufficient achievement because their goal is simply to gain control over and analyze a large, mostly undocumented code base. That certainly has significant value. But, for others, this is just a step toward revising the architecture of their legacy application. The three most common goals in such transformation are:

- Separation of UI and BL, usually with the goal of replacing the UI;
- Evolution toward OERA or similar layered architecture; and
- Evolution toward a Service Oriented Architecture.

Needless to say, these are not mutually exclusive goals. I have some ideas I am working on related to service identification which should be applicable without major architectural shifts. I also have some ideas on providing data layer separation which might also be possible without major changes. UI separation is definitely a harder problem, but a real hot button, so it is going to get a lot of my attention.

I will note that most UML OO development starts with a model which is highly abstracted from the actual code and then resolves toward a progressively more concrete and specific structure. The need here is to go in the opposite direction, which is a substantially harder proposition and one on which far less work has been done ... but I have ideas!



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 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 being 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





Computing Integrity, Inc.

<http://www.cintegrity.com>

thomas@cintegrity.com

510-233-5400

