Can the CDT DOM be used to represent languages other than C and C++?

Doug Schaefer, senior software developer, Eclipse CDT project lead
QNX Software Systems
dschaefer@qnx.com

In a lot of ways, the CDT is already a multi-language framework. The first misconception we had to get over when we started our parser work in the CDT was that C++ is a superset of C. This is actually not 100% true. In C, you can call a function without having it declared in scope. There is also scoping weirdness with C structs where the scope is actually flattened. There are also a few extra keywords in C99 that haven't been added to C++. These differences are minor but lead us down the path where we actually have separate, yet related, parsers for these two languages. We also found some good performance gains for C parsing by not having to check for all of the additional constructs that have been added to C++.

Along with the separate parsers for the C and C++, we have different representations in the DOM for these two languages. Since they are so related, we have a large number of interfaces that are common between the two, but we have allowed for language specifics through interface extensions. As well, most of the common interfaces have different implementations for C versus C++ to allow for the extensions to get hooked up.

So, given that we already handle essentially two languages, it begs the question: Can the CDT DOM handle other languages than C and C++? We haven't really tried but my cursory investigation and my gut tells me that, yet, we can. There are certain elements common to C and C++ that are common to many other languages as well. This position paper will discuss what these elements are and show the path we need to go down to confirm whether this is true.

The CDT DOM

What is a DOM anyway? Well, our interpretation of DOM is similar to the traditional XML DOM definition. It is an object model that describes the structure of some source text. Ideally, you can make changes to the object model and have those changes automatically reflected in the source text or serialized out to some new source text.

For XML, using a package like xerces, and for Java, using the JDT's DOM, this actually works well. For C and C++ this is actually a quite a bit more difficult since the source text may actually be spread across multiple files thanks to our good friend the C preprocessor. But, we've given it a good try and we will see down the road how successful a writable CDT DOM will be.

For the main part, the CDT DOM in it's current state, is a read-only representation of a C or C++ translation unit. The representation is created using a scanner, which tokenizes the program text and executes C preprocessor commands, and a parser which takes the tokens and creates an Abstract Syntax Tree (AST) that represents the structure of the text. The AST contains location information that maps AST nodes through the various preprocessor translations, through macros and inclusions, back to the source text in files. As well, it is possible to navigate the DOM from references to declarations using Binding information (the origin of the term is the JDT DOM and we couldn't think of anything better). The Binding information generally represents semantic or logical elements such as types, variables, functions.

The key elements that hold the DOM together are Names and Bindings. As I mentioned Bindings refer to semantic or logical elements in a program. These are the things that actually represent data store, functions that change the data store, and types that describe both of these. Names are program text used to declare and refer to these semantic elements.

Being able to handle Names and Bindings is the key to some of the editor and source navigation features of the CDT. For example, the implementation of open declaration starts with a source text selection, looks up to find the most appropriate Name contained in the selection, gets the Binding that this name refers too, and then searches for the most appropriate Name that represents the declaration of that Binding that we want to navigate to. Content assist works similarly, by treating the identifier preceding the cursor as a Name prefix and then finding the most appropriate Bindings that match that prefix.

The potential uses of the CDT DOM are only limited by imagination. As we evolve the DOM to make it writable/serializable we can support refactoring, beyond the rename refactoring we already have. There are also numerous things people can build in the code analysis/code generation area.

Multi-Language Support

I have always been interested in programming languages, the most important tools used by software developers. Multiple language exist to support different programming paradigms which themselves are designed to support different problem domains. The most common languages are general purpose and can be used to build solve most computational problems. What usually differentiates them are certain features that make it easier to express a certain solution. For example C++ classes make it easier to represent and enforce encapsulate types than you could in C.

One thing I see in common with most programming languages is the same thing that makes holds the CDT DOM together, i.e. the use of Names to represent logical program elements. If we can make the DOM easily extendible to other languages, then it should be possible to easily adapt the CDT editor and navigation features that use the DOM to these languages.