www.iar.com

Master's Thesis proposal: Modelling Option Logic in an IDE

Background

IAR Embedded Workbench is a C/C++ Integrated Development Environment (IDE). As such, one of the primary tasks it performs is to compile C/C++ source code into executable code. To compile a project correctly, there are a number of different configuration options the user might need to set (for example, optimization levels, language compliance level, CPU variant to generate code for, etc). These *abstract* options are then translated into *concrete* options (typically command line switches to a compiler or linker executable). The set of options and the associated logic is sometimes referred to as the *project model.*

The project model is used not only by the IAR Embedded Workbench IDE application. For example, IAR Systems supports Eclipse as an alternate IDE for the company's compilers and debuggers. This means that the project model needs to be accessible from Eclipse as well.

In many cases, the translation from the "input domain" (abstract options) to the "output domain" (the concrete command line switches) is straightforward. For example, there might be a switch that instructs the compiler to generate *debug information* which translates to a single command line switch (-g) to the compiler.

Other options are more complicated. For example, there are two options used for selecting what entry point should be used: one to override the default, and the other to specify the name of the symbol to use instead. The latter option is not relevant if the first is not enabled, and the latter should refer to a symbol if the former is enabled. If we just pass "-e" followed by an empty string to the compiler, we take the risk of getting error messages that are difficult for the user to understand.

These relationships between options can easily become arbitrarily complex; relevant restrictions must be defined. For example, preventing the user from specifying an unknown symbol as entry point to the program is not feasible, because the option system has no way of reliably knowing the set of defined symbols.

The task

The task of this thesis is to develop a way to express the option logic as an abstract model that can be integrated into generic implementations, both IDEs and in standalone tools.

We suggest implementing a prototype—project file transformation tool—which can transform project files according to a set of rules. The tool has the following specifications:

- It takes the following inputs
  - The option model: a file that lists the available options, and the possible values of the options, together with a specification of the option logic (how value changes to one option affect other options)
  - The project file which lists the "current value" of one or more of these options.
  - A single option-value pair specifying a change to a single option to perform.
- It produces the following outputs
  - A list of changes which need to be made to the project file to apply the given change, or an error message if the change cannot be applied (if it for example leads to inconsistent values).
  - A new project file with the applied change.

The prototype should address some of the weaknesses that the current model used in IAR Embedded Workbench has:

1. The relationships between options are implicitly encoded in C++ source code: the fact that a given option depends on another option can only be discovered by analyzing the C++ source code itself. Also, any effects an option has on other options can only be detected by actually running the code.
2. The project model is difficult to access from other clients (e.g. Eclipse), because the option logic is hardwired in C++.
3. The code that implements the option logic cannot be executed outside of the IDE dialog box of the corresponding UI controls and thus cannot be reliably tested.
4. The set of available options cannot be queried or inspected from outside the IDE itself.
5. The amount of code necessary to add support for a new option is larger than desirable.

Questions

1. Is there any existing literature on this subject?
2. Is it possible to specify option logic in such a way that we can
   a) reduce the amount of code necessary compared to the current approach?
   b) reuse common option definitions between different build targets?
   c) improve testability by allowing option logic to be executed in a controlled environment?

   d) access the option logic from different execution environments (e.g. Windows, Eclipse)?

3. What are the trade-offs involved, and how do they affect how the option logic can be expressed?

Contact at IAR Systems

Susanne Dahlén *Development Director*
IAR Systems AB
Box 23051, Strandbodgatan 1
SE-750 23 Uppsala, SWEDEN
Mobile: +46 708 66 10 76 Phone: +46 18 16 78 00 Fax: +46 18 16 78 01
E-mail: susanne.dahlen@iar.com Website: www.iar.com
Twitter: www.twitter.com/iarsystems