DylanWorks Edit/Compile/Debug Model

Scott McKay & Roger Jarrett

1. Introduction

The paper describes the DylanWorks model for the edit/compile/debug cycle. It describes both the "interactive" (incremental compilation and dynamic linking) model and the "static" model (production of an application. It also describes how the edit/compile/debug cycle fits in with the DylanWorks system configuration management and patching modules.

2. The Interactive Cycle

Assuming that there is an existing Dylan project, select a project to work on and an access path to a "remote" platform. Selecting a project loads up the project definition (and the derived info stuff) into DylanWorks, and might download the actual code into the remote host.

Blah blah blah...

Two possible schemes:

- To start things up, you malloc a bunch of memory and then load object files into it. Pros: good control over the process. Cons: you have to write a loader; there's a discontinuity between the interactive process and the delivery process.
- To start things up, you link up a .EXE file with the "spy" and "debug" modules in it, then start it up in the debugger. Pros: you can use the stuff supplied by the OS; no discontinuity. Cons: a bit trickier to do the dynamic linking, maybe.

The consensus is that the second scheme is better.

3. Building an Application for Delivery

Blah blah blah...

4. Patching

- Dynamic patching via IDVM, as in Lisp
- "Static" patching by recompiling explicitly changed definitions, recompiling affected functions (determined via dependency tracking), regenerating object files, and relinking.

Blah blah blah...

At Harlequin created on October 11, 1994 and last modified at Harlequin on January 3, 1996.