Dyninst and DPCL:

API Integration

James Waskiewicz
Overview

- DPCL in a Nutshell
- Some History of the API Evolution
- Goals of API Integration
- Core Differences
- A Working Hybrid
- Proposed Merging Strategy
- Conclusion
DPCL in a Nutshell

- DPCL is a Unix/TCP socket based API infrastructure for
  - The organization of dynamic instrumentation (mutator) processes across multiple nodes
  - Reporting the information generated back to the core application

- Based on the Dyninst API

- The DPCL API permits the development of analysis tools on a high level by encapsulating
  - Dynamic instrumentation
  - A Distributed infrastructure

- Supported OS’s: AIX, Linux
Some History of the API Evolution

- DPCL began as an IBM project in 1997
  - Licensed DyninstAPI.
  - Originally only supported on AIX
- In the course of development, libIbmBPatch was written
  - Essentially a heavily modified version of the DyninstAPI
  - Intended to support multi-threading
  - Exports a DyninstAPI veneer
- The DPCL-DyninstAPI was expanded/altered
Some History, Con’t

- **Major DPCL project strategy shift:**
  - On March 14, 2001 at 2:14 PM EST, the DPCL source was opened
  - Linux port unveiled

- **Byproduct/Upshot**
  - Introduction of a variant of the DyninstAPI to the open-source world
  - Mass confusion in the streets, rioting, etc.

- **Something MUST be done!**
Goals of API Integration

- **DPCL on any Dyninst-supported Platform**
  - IBM currently only supports AIX and Linux
  - Dyninst supports AIX (RS6k), Linux and Windows (i386), SGI (MIPS), Solaris (Sparc & i386), Windows CE, and OSF (Alpha)
  - Upcoming ports include IA64 Linux
  - Goal is a DPCL Infrastructure that exclusively relies on Dyninst for instrumentation

- **A Concern**
  - Merely adding all of IBM’s modifications to Dyninst would create a bloated API
  - Many differences are small semantic issues
Core Differences

- **Minor Syntactic (largest group)**
  
  - Eg: `someClass->getLastError()` vs. `someClass->getLastErrorCode()`

- **Major Syntactic**
  
  - DPCL uses STL, Dyninst has custom template container classes
  
  - DPCL has added some functions that qualify as non-trivial API extensions: eg. The notion of inclusive and exclusive points; referring to a mutatee function call with a variableExpr reference
  
  - Callback Function prototype differences:
    
    - Dyninst has many types of callbacks, eg. BPatchExitCallback, BPatchSignalCallback...
    
    - DPCL registers all callbacks as BPatchThreadEventCallback
Core Differences, con’t.

- **Minor Functional**
  - Eg. Char *getMangledName() vs. the pair:
    int getMangledName(char *buf, int sz); and
    int getMangledNameLength()
  - DPCL Daemon processes need to be fixed to properly inherit Dyninst environment variables.

- **Major Functional**
  - DPCL Daemon expects a shared memory API to be loaded into a mutatee upon attaching to it.
  - AIX Multithread support
A Working Hybrid (AIX)

- ** Modifications to DPCL Source Tree 
  - Disabling the IBM version of Dyninst can be done with a compiler flag
    - Many compilation errors result
  - Dyninst environment variables explicitly supplied in certain places
  - DPCL Daemon explicitly modified to load DPCL shared memory API
  - Rewrote some code segments to properly use a slightly expanded version of DyninstAPI
    - Expanded to use STL vector/algorithm
    - Functions added/modified to take Core Differences into account
A Working Hybrid (AIX), Con’t

- **Compile DPCL w/g++, not xlC**
  - For consistent C++ name mangling
  - Probably easier than getting Dyninst to compile with xlC
  - Many small modifications needed in the configure/make scripts
  - Some xlC specific notions, eg. importing and exporting code-objects must be reworked into a gnu shared library, or explicitly specified dynamic object.
A Working Hybrid (Linux)

- This page intentionally left blank
Proposed Merging Strategy

- **Working with the DPCL developers**
  - Currently in a discussion over how core differences should be addressed.
  - Each side should make some changes

- **Implementation of Changes**
  - We expect several small changes to the API in this process with some larger extensions
  - Relatively small user base can most likely “deal with it”
  - Use a compiler flag to handle forwards/backwards compatibility for a transition period.
Proposed Merging Strategy, Con’t

- **Dyninst API modification issues**
  - Too many names for the same thing impedes clarity, and thus utility
  - Need to keep dyninstAPI “reasonable”
    • Make sensible additions/modifications to the API to support the needs of DPCL
    • Implement a formalized naming convention to ensure consistency
    • Avoid API bloat, but make sure it’s still “comfortable” to use
Conclusions

- The APIs are not so different as to make this problem intractable
- Once we decide where the API boundary lies, implementation will not take very long
- Consider development of a general DPCL test suite for use in further hybrid porting efforts