PRISMTECH
Productivity Tools & Middleware

Experience Report
CORBA brings new life to legacy application

Ada Europe
June 2006

Introduction

- Context
  - Existing legacy Ada83 system (Combat Management System)
- Need
  - Add new features
  - Integrate in a more open environment
- Question
  - Keep Ada ?
    - Move to Ada 95 or not ?
  - Opportunity to move to a different programming language ?

Key issues of the combat management system

- Existing system
  - HpRt + HpUX
  - Ada83
  - Proprietary middleware
  - High performance of tranport used in legacy middleware
- Requirements for upgraded system
  - Linux + Windows
  - Ada95
  - Keep proprietary middleware compatability
  - system performance must not degrade
  - add an external Java GUI to access large data structures
  - use COTS as often as possible

Technical issues

- CORBA recognized as a standard solution
  - COTS available for Ada/Java and various platforms
  - Existing OMG mapping for Ada/Java and even C++, if required later, provide a natural interface between languages
- data structures to share were huge
  - Converting types between legacy and CORBA world should be avoided
- legacy system was based on a large ada types dictionnary
  - focus on typing
  - try to maintain advantage of previous validation
  - try to keep automatic code generation

Creation of a « rich » IDL

- Try to express Ada types through CORBA IDL
- Most Ada types could find an obvious IDL equivalent, including packages transposed to IDL modules.
- Some Ada constraints are no equivalence in IDL
  - Subtypes, range …
  - introduce pragmas in IDL to keep this kind of information in IDL
- « ada2idl » tool developped to automate the process
  - Based on Top Graph'X compilation technology
  - Automatically converted dozens of Ada packages
  - Makes iterations easy (processus additions or modifications).

Ada to IDL generation

- Allow or idl2ada compiler to understand the introdced pragmas, so that the generated Ada packages wold embed all the original typing details

IDL versus original Ada

- Ada specification

```
package X is

   type T_Short is range – 32768 ..
     32767 ;


   type Percent is new T_Short range
     0 .. 100 ;


   subtype Another_Short is T_Short
     range 0 .. 1000 ;


end X;
```

- **IDL generated**

```
module X
{
typedef short T_SHORT ;
#pragma OrbAda_Directive "Range" "
     T_SHORT  -32768    32767"

typedef T_SHORT   PERCENT ;
#pragma OrbAda_Directive "Range" "
     PERCENT  0  100"

#pragma OrbAda_Directive
     "Insert_Line" "   subtype
     ANOTHER_SHORT is T_SHORT
     range 0 .. 1000 ;"
} ;
```

IDL to Ada generation

- Standard mecanism
- Will respect Ada95 mapping defined by OMG

Original Ada versus regenerated Ada

- Original Ada

```
package X is
   type T_Short is range - 32768 ..
     32767 ;

   type Percent is new T_Short range
     0 .. 100 ;

   subtype Another_Short is T_Short
     range 0 .. 1000 ;

end X;
```

- Regenerated Ada

```
-- From IDL file X.idl
with Corba_Ios ;
with Corba.Object ;
package X is
type T_SHORT is new Corba.Short
     range -32768 .. 32767 ;
   for T_SHORT'size use 16;

   type PERCENT is new T_SHORT
     range 0 .. 100 ;
   for PERCENT'size use 16;

   subtype ANOTHER_SHORT is
     T_SHORT range 0 .. 1000 ;

end X;
```

Automated controls

- Test program automatically generated
  - Will be compiled and run twice
    - With legacy system types packages
    - With newly generated types packages
  - Output comparison will ensure that new types have correct properties to ensure correct behavior of the system
    - types sizes controlled
    - ranges controlled

Application rebuilt

- Original source nearly ready to compile
  - array types using enumerated types as index could not be expressed in IDL
    - IDL arrays used
    - Adapt source code, using ‘pos and ‘val
    - Adaptations accepted as safe enough since original code was validated
    - Very small number of modifications
- Recompiled application runs fine
- Newly regenerated system is « CORBA ready »

Adding an interface

- New system still does not support any CORBA interface
- Define IDL to link combat system to Java GUI :

```
Module GUI
{
  update (Track);
} ;

Module Combat_System
{
  register (GUI);
  modify (Track);
} ;
```

The middleware main loops

- 2 solutions provided by CORBA
  - ORB.run()
    - Blocks current thread until ORB.shutdown()
    - Dispatch incoming requests to 1 or several execution threads
    - Preferred solution for newly designed system
    - Could be used in the combat system case
  - ORB.work_pending() + ORB.process_work()
    - Polling method
    - Often used in combination with X11 Xt event processing mecanism
    - Often preferred solution when adding CORBA to old systems

Use of the « rich » IDL by Java/C++ environments

- Introduced pragmas not recognized by foreign IDL compilers
  - Pragmas just ignored => standard IDL.
- Top Graph'X idl2java and idl2cpp compilers can be extended to generate additional controls

Robustness added to Java/C++ callers

- Unexpected but interesting side effect
  - Incorrectly initialized parameters sent from C++ or Java
    - Generates Constraint_Error exception in Ada code
      - *Generally concerns code generated by idl2ada compiler before calling application method implementation*
    - Exception caught by CORBA library and transformed to an internal CORBA exception
    - CORBA exception returned to caller

Key points of the migration

- Develop new tools such as an Ada to IDL compiler to automatically generate type safe IDL for existing Ada interfaces
- Extend CORBA IDL to more closely support Ada type system, this will remove the need for inefficient data conversions at runtime, maintaining system performance and allowing legacy Ada interfaces to remain unchanged
- Add CORBA bindings supporting existing Ada type system
- Automatic code generation techniques made the task of migrating a huge amont of legacy code dramatically more efficient
- Being able to support the legacy Ada code without changes meant that any previous testing was still valid

Possible extensions and comments

- Ada2idl tool capabilities to be extended
  - Needs to be adapted to more complex cases
  - Strategy might need to ba adapted to environment constraints
- Original Ada types generated for the combat system
  - Allows direct « rich » IDL generation
- Rich IDL is not standard …
  - But makes life easiear with legacy systems
  - Is still compliant with standard IDL

Middleware cohabitation

- Different contexts and constraints lead to different approach
- PrismTech/Top Graph'X involved in large ATC system (DSNA/STNA)
  - Validated modules cannot be modified and re-delivered
    - Previous solution strictly forbidden
  - New system will be built on top of CORBA/DDS
    - CCM compliant (Cardamom)
    - ICOM (STNA) to talk with Corba/DDS
    - Develop wrappers to allow middleware co-existence