Creating CORBA Applications Using UML and SDL

Morgan Björkander
mbj@telelogic.com
Two Observations

• Many profiles are being defined to tailor the UML towards specific domains
• There is an increased focus on specifying behavior in UML
  – Making it executable
  – Allowing early verification
CORBA IDL—an Abstract Language

• IDL is used as an abstract language in the sense that it is target language independent
  – What if it were possible to specify behavior in a manner that is also target language independent?
• UML contains a profile mechanism that can be used to tailor it towards specific domains, but also to create language mappings
Creating a CORBA Application

Java ORB

IDL to Java Compiler

Client Developer

app.java

stubs.java

Java ORB Run-time Lib

ClientExecutable

IDL Source

IDL Developer

IDL to C++ Compiler

Server Developer

types.hh

stubs.cc

impl.cc

serv.hh

skels.cc

C++ ORB Run-time Lib

ServerExecutable

C++ ORB

Creating CORBA Applications Using UML and SDL

© Telelogic 2000
Alternative Approach

Client Developer

Server Developer

Modeling Language Application

Java ORB

IDL to Java Compiler

app.java

stubs.java

Java ORB Run-time Lib

C++ ORB

IDL to C++ Compiler

types.hh

stubs.cc

impl.cc

serv.hh

skels.cc

C++ ORB Run-time Lib

Client Executable

Server Executable

Executable
Raising the Abstraction Level

**Target language independence**

**Platform independence**

**ORB**

TCP/IP
Process Considerations

- Create Model
- Verify Behavior
- Specify Deployment
- Generate Code
- Implement Behavior
- Debug
- Create IDL
- Generate Code
Importing and Exporting IDL

• Take advantage of already existing clients and servers in the network
  – Import IDL exposed by external servers
  – Export IDL to be included by external clients
UML Profiles

• Stereotypes
  – Model elements that can be used as if they are a part of UML
  – Specialize existing model elements

• Tagged Values
  – Define additional stereotype properties

• Constraints
  – Restrict the use of stereotypes
The UML Profile for CORBA

- Specifies how to use interface descriptions (IDL) in UML
- Does not (and was not intended to) cover the logical next step: behavior

// IDL sample code
module CCS {
    interface Thermometer {
        readonly attribute short temperature;
    };
    interface Thermostat : Thermometer {
        short get_nominal();
        short set_nominal(in short new_temp);
    };
};
Action Semantics for the UML

• Turn the UML into an executable language
  – Provides the semantics of actions
    • Lacks a notation
    • Does not include a data model

• Necessary to map a language on top of the action semantics (such as C++, Java, or SDL)
  – Target language independence?
  – At some point in the future, UML may provide sufficient detail by itself
SDL-2000

The Specification and Description Language

- SDL is an executable modeling language that is standardized by the ITU
  - Similar to UML in many ways, but specifically geared towards modeling of event-driven, distributed applications

- It is often used as a full-fledged programming language
  - There are many indications that UML is heading in the same direction
Using IDL

- Incorporate IDL data types in the action language
  - Basic data types
  - User-defined types
  - Extend types with operators defining “+”, “-”, type casting rules, etc.

- Attributes and parameters can be declared using these IDL data types
  - IDL interfaces may be shown textually or graphically (using the UML profile for CORBA)
Designing the Application

- **Agents** define logical components
  - define the system’s functionality

- Bi-directional interfaces specify contracts between agents
  - required interfaces
  - implemented interfaces
  - an agent may implement multiple interfaces

```
Station

servant acts as client

Relay

servant acts as server

Receiver

Radio

```

© Telelogic 2000
An Example: Static Structure

- Describe the classes used in the system
  - agents are stereotypes of classes
- Each agent may have structure and behavior
An Example: IDL Interfaces

// Customer’s view of accounts
interface iAccount {
    typedef double MoneyT;
    readonly attribute MoneyT balance;
    void insert(in MoneyT amount);
    void withdraw(in MoneyT amount)
        raises (InsufficientBalance);
};

// Manager’s view of accounts
interface iAccManager : iAccount {
    iAccount addInterest(in float rate);
};

// Customer’s view of cashiers
interface iCashier {
    iAccount createAccount(in string code, out number);
    void terminateAccount(in iAccount account)
        raises (NoAccount, AccountLocked);
    iAccount loginAccount(in string code, in long number)
        raises (WrongCode, WrongNumber);
};
An Example: Dynamic Structure

- Describe how agents interact with each other
  - connections
  - dynamic creation
- Provide initialization information
  - number of instances to be created at start-up
    - multiplicity
  - compare with main function
An Example: Behavior

- Action semantics defines the behavior of
  - operation bodies
  - transitions
  - entry/exit actions
- Each such computational unit is represented by a *procedure*
  - a sequence of actions

```
DCL /* attributes of agent */
private number long := 0;
```

```
DCL /* local procedure variables */
acc iAccount;
```

```
iAccount createAccount (in string code, out long number)
```

```
acc := create Account (code)
```

```
Acc != NULL
```

```
[number := number +1]
```

```
storeAccount (acc, number)
```

```
InvalidReference
```

```
account
```
An Example: Using CORBA Services

DCL

inc CosNaming::NamingContext, /* initial naming context */
name CosNaming::Name, /* Name to resolve */
db iDatabase;

name[1].id := ‘corporate’
name[2].id := ‘develop’
name[3].id := ‘database’

db :=
inc.resolve (name)

db != NULL

NotFound
CannotProceed
InvalidName
Deployment

- Determine if and how to partition the system
- Determine code generation policies for each partition
  - orb
  - target language
- Manage the setting of POA policies
- Tagged values may be used to control desired behavior, such as:
  - persistent objects
  - transient objects
Application Code Generation

- The final step is to generate the application code for the different parts of the system according to deployment information
- Apply the IDL mapping rules for the target language
- Generate implementation specific details according to model
References

- OMG: The UML Profile for CORBA, 2000
- OMG: Action Semantics for the UML, RFP, 1998
- OMG: Action Semantics for the UML, Joint Initial Submission, 2000