

The Ada Distributed Systems Annex

FOSDEM 2009

Thomas Quinot

AdaCore

Brussels, 2009-02-07

- 1 Introduction
- 2 A simple "zeitgeist" application
- 3 Social networking made easy
- 4 Distributed Ada
- 5 Implementation of the DSA
- 6 Conclusion

Outline

- 1 Introduction
- 2 A simple "zeitgeist" application
- 3 Social networking made easy
- 4 Distributed Ada
- 5 Implementation of the DSA
- 6 Conclusion

Motivation for distribution

Many aspects of software engineering require, or can benefit from, distributed technology:

- Load balancing
- Fault tolerance
- Interconnection between multiple agents
- ...

Distribution models

A distributed application design relies on the abstractions of a distribution model to express the interactions between application components:

Message passing

Non-structured

Remote subprogram calls

Based on natural abstraction boundaries (subprograms)

Distributed objects

Extend RPC to object-oriented design

(+ distributed shared memory)

Distribution models and monolithic programming models

Parallel between distribution models and monolithic programming models:

GOTO
↓
Structured
↓
Object-oriented

Message passing
↓
RPC
↓
Distributed objects

Distributed programming

Communication APIs → BSD sockets

Cumbersome, error-prone

Specialized distribution APIs → MPI, CORBA

Intrusive, steep learning curve, freeze distribution boundaries

Distributed language → Ada DSA, Java RMI, Modula 3

Seamless integration in application, extend languages' abstraction to support distribution.

Given a distribution model and an implementation (*middleware*), how to handle interoperability with other distributed components?

Distribution in Ada 95/2005

- Ada has built-in features for contract-based programming
- Natural extension to distribution
- Comparable to Modula/3 Network Objects, Java Remote Method Invocation.

Outline

- 1 Introduction
- 2 A simple "zeitgeist" application**
- 3 Social networking made easy
- 4 Distributed Ada
- 5 Implementation of the DSA
- 6 Conclusion

Service definition

```
package Zeitgeist is
  type News_Item is
    Author, Message : Unbounded_String;
  end News_Item;
  type News_Items is array (Positive range <>)
    of News_Item;

  procedure Post (Item : News_Item);
  function Whats_Up return News_Items;
end Zeitgeist;
```

Making it remote

We want to make calls to the service from another application

```
package Zeitgeist is
  pragma Remote_Call_Interface;
  type News_Item is
    Author, Message : Unbounded_String;
  end News_Item;
  type News_Items is array (Positive range <>)
    of News_Item;

  procedure Post (Item : News_Item);
  function Whats_Up return News_Items;
end Zeitgeist;
```

Building the complete application

Once distribution pragmas have been added to identify *possible* distribution boundaries, the user can:

- build a monolithic application as usual
- split units according to distribution boundaries \Rightarrow **partitioning**

The partitioning process is implementation defined.

Partitioning configuration for a minimal application

```
configuration Dist_App is  
  pragma Starter (None);  
  — User starts each partition manually  
  
  ServerP : Partition := (Zeitgeist);  
  — RCI package Zeitgeist is on partition ServerP  
  ClientP : Partition := ();  
  — Partition ClientP has no RCI packages  
  
  for ClientP 'Termination use Local_Termination;  
  — No global termination  
  
  procedure Server_Main is in ServerP;  
  — Main subprogram of master partition  
  procedure Client_Main;  
  for ClientP 'Main use Client_Main;  
  — Main subprogram of slave partition  
end Dist_App;
```

Outline

- 1 Introduction
- 2 A simple "zeitgeist" application
- 3 Social networking made easy**
- 4 Distributed Ada
- 5 Implementation of the DSA
- 6 Conclusion

Components

You have a shared bulletin board — now what if you want direct messaging between users?

```
package Chat_Users is  
  type User is abstract tagged limited private ;  
  type User_Ref is access all User'Class ;  
  function Name (Who : User) return String ;  
  procedure Say  
    (From      : String ;  
     To_Whom   : User ;  
     What      : String ) ;  
end Chat_Users ;
```

Communicating components

OO provides flexible object interactions — now let's extend these across partition boundaries!

```
package Chat_Users is
```

```
  pragma Remote_Types;
```

```
  type User is abstract tagged limited private;
```

```
  type User_Ref is access all User'Class;
```

```
  — Remote Access to Class-Wide type
```

```
  function Name (Who : User) return String;
```

```
  procedure Say
```

```
    (From      : String;
```

```
     To_Whom   : User;
```

```
     What      : String);
```

```
end Chat_Users;
```

A Remote Access to Class-Wide type may reference *remote* objects.

Getting in touch

Each partition creates objects, and makes RACW that point to these object.
Now how do you initially obtain these pointers to remote objects?

```
with Chat_Users; use Chat_Users;  
package Zeitgeist is  
  pragma Remote_Call_Interface;  
  type News_Item is  
    Author : User_Ref;  
    Message : Unbounded_String;  
end News_Item;  
  type News_Items is array (Positive range <>)  
    of News_Item;  
  
  procedure Post (Item : News_Item);  
  function Whats_Up return News_Items; end Zeitgeist;
```

RCIs allow initially passing around RACW values.

Outline

- 1 Introduction
- 2 A simple "zeitgeist" application
- 3 Social networking made easy
- 4 Distributed Ada**
- 5 Implementation of the DSA
- 6 Conclusion

Unit categories and categorization pragmas

Pure

Base data types declarations (no named access types, no indirection)

No internal state

Remote_Types

Complex data types (possibly encapsulating hidden access types), remote access types

Remote_Call_Interface

Remotely callable subprograms

Semantic dependency (*with*) constraints:

Pure ← Remote_Types ← Remote_Call_Interface ← non-categorized

Remote subprograms

Formal parameters must "support external streaming":

- no access types
- no limited types

... unless *stream attributes* are specified.

```
package Streamable_Lists is  
  type List is private;  
  procedure My_Read  
    (S : access Ada.Streams.Root_Stream_Type'Class;  
     V : out List);  
  procedure My_Write  
    (S : access Ada.Streams.Root_Stream_Type'Class;  
     V : List);  
  for List'Read use My_Read;  
  for List'Write use My_Write;  
private  
  ...  
end Streamable_Lists;
```

Remote access types

Access types declared in the visible part of an RT or RCI unit.

Remote access to subprogram types may designate remote subprograms (subprograms declared in the visible part of a remote call interface unit).

Remote access to class-wide types may designate "suitable" tagged types (limited private types declared in the visible part of a declared pure or remote types unit).

Behind the scene

For each remotely callable subprogram (RCI subprogram, primitive operation of RACW type), the compiler generates:

Calling stubs (client side)

Marshall arguments, make remote call, unmarshall result;

Receiving stubs (server side)

Unmarshall arguments, make upcall to application code, marshall result (or exception).

Generated code makes calls to supporting routines provided by the PCS.

Outline

- 1 Introduction
- 2 A simple "zeitgeist" application
- 3 Social networking made easy
- 4 Distributed Ada
- 5 Implementation of the DSA**
- 6 Conclusion

Anatomy of a DSA implementation

Code generator

Produces client and server stubs. Part of the compiler.

Partitioning tool

Assigns units to partitions and builds executables.

Partition communication subsystem (PCS)

Runtime library called by the generated code, providing the communication services.

DSA support in GNAT

DSA code generation is an integral part of the GNAT compiler.

PCS + partitioning tool (gnatdist) bundles:

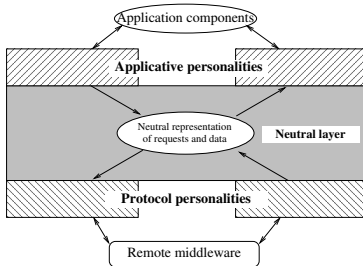
GLADE (GARLIC PCS) Legacy implementation using an ad hoc, specific protocol

PolyORB/DSA (PolyORB PCS) Based on reusable polymorphic middleware, can use a variety of standard protocols (GIOP, SOAP...) to allow interoperability with other distribution models such as CORBA.

- Available from <http://libre.adacore.com/>.
- Commercial support enquiries to sales@adacore.com.

PolyORB: a polymorphic, reusable middleware architecture

- A **generic, configurable, interoperable** middleware for **critical** distributed applications.
- Highly decoupled, modular architecture, customizable according to application and environment requirements, providing maximal interoperability across distribution models.
- Formally proven middleware kernel: the μ Broker.



Outline

- 1 Introduction
- 2 A simple "zeitgeist" application
- 3 Social networking made easy
- 4 Distributed Ada
- 5 Implementation of the DSA
- 6 Conclusion

Ada DSA summary

- DSA (Annex E) incorporates distribution within the language, preserving the language's abstraction and strong typing features.
- No new API, natural extension of contract-based programming paradigm.
- PolyORB/DSA allows interoperability with CORBA, SOAP systems.
- GNAT implementation freely available from <http://libre.adacore.com/>.
- For commercial support contact sales@adacore.com.

Questions?

?