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
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
- ...
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:

- 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
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;
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;
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 → partitioning

The partitioning process is implementation defined.
Partitioning configuration for a minimal application

```ada
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;
```

Thomas Quinot
The Ada Distributed Systems Annex
Introduction
A simple "zeitgeist" application
Social networking made easy
Distributed Ada
Implementation of the DSA
Conclusion

1. Introduction
2. A simple "zeitgeist" application
3. Social networking made easy
4. Distributed Ada
5. Implementation of the DSA
6. Conclusion
You have a shared bulletin board — now what if you want direct messaging between users?

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

```ada
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.

```ada
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)**
- Marshal arguments, make remote call, unmarshal result;

**Receiving stubs (server side)**
- Unmarshal arguments, make upcall to application code, marshal result (or exception).

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

- 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 \( \mu \)Broker.

![Diagram showing the architecture of PolyORB with application components, neutral layer, applicative personalities, protocol personalities, and remote middleware connections.]
Outline

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


For commercial support contact sales@adacore.com.
Questions?