The Ocarina Tool Suite

Thomas Vergnaud <thomas.vergnaud@enst.fr>
Motivation

• ENST is developing a middleware architecture: PolyORB
  • generic, configurable, interoperable
  • enables middleware verification

• create a tool chain to configure PolyORB
  • lightweight
  • modular

• automatic deployment of distributed applications
  • configure the distribution infrastructure according to the application
  • ensure reliability of the runtime
  • facilitates verifications on the whole application
The Ocarina Tool Set

- a set of tools to describe, generate and deploy distributed applications

- editors
  - text: Emacs
  - graphical: Dia

- central core
  - general semantic verifications
  - to be used in existing applications to provide AADL capabilities
    - Cheddar (univ. Brest): schedulability analysis
    - GLADE (AdaCore/ENST): implementation of the distributed systems annex for Ada 95

- several I/O modules
  - text
  - graphical syntax (Dia)
  - XML

- code generator: Gaia
Gaia: rationale

- to generate and deploy distributed applications

- need for an AADL runtime
  - handle the communications
  - manage the AADL threads

- some issues:
  - communications must be correctly handled
  - they should match to the AADL description
  - application elements should not be able to take control of the system execution
Gaia: general concepts

- A runtime structure is created from the AADL description
  - configured from the AADL description

- Application elements are encapsulated and driven by the runtime
  - do not control the runtime
  - do not have visibility on other parts of the application
Architectural entities

- execution platform components: provide information on deployment
  - machines
  - networks
  - …
- software components: model the applications that must be generated
  - processes: model a logical node (partition)
  - threads: active elements of the applications
  - subprograms: passive elements
  - data: data structures
- systems
  - non functional elements

- threads symbolize the runtime
- subprograms describe the application
- systems do not correspond to anything in the application
AADL → programming language

- depend **only** on the target language
  - no relation with the runtime
- generate wrappers from AADL description
- generate data types from AADL data components

```ada
data dt

... end dt;

subprogram sp
features
  a : in parameter dt;
  b : out parameter dt;
properties
  source_language => Ada;
  source_name => "pack";
end s;

procedure sp (a : in dt; b : out dt)
  is
    begin
      pack.sp (a, b);
    end sp;
```

```ada
type dt is...
```
• data components
  • translation from AADL to programming language
    • *unique AADL representation, many language translations*
  • subcomponents are fields of records
  • GAIA::data_type : enumeration
    • *integer, float, string, boolean, etc.*
    • *provide the required semantics to generate data type code using a programming language*
    • *the use of properties help manage data component extension*

• subprograms
  • if source code is specified => generate a wrapper for the specified language
  • if call sequences are provided => generate the sequences
  • if nothing is provided => generate an empty procedure
Distribution models

• message passing
  • event data ports associated to threads
• remote procedure call
  • subprograms as features of a thread
  • subprogram access provided by a thread and required by a subprogram ran by another thread (soon)
• distributed object
  • data access provided by a thread; the data component provides subprograms, the same way as for remote procedure calls
• shared memory
  • access to data component provided by a thread

• threads are crucial, since they materialize the runtime
Which runtime for the application?

• AADL runtime = middleware
  • “high-level” runtime
  • can handle all distribution paradigms
  • need for an adaptation layer generated from the AADL thread descriptions

• AADL runtime = minimal runtime
  • “low-level” runtime
  • scheduler + sockets (+ operating system?)
  • requires additional AADL components to describe the management of communications
  • requires AADL threads to be expanded
PolyORB

- implementation of the schizophrenic middleware architecture
- 3-layer structure
  - applicative personalities: CORBA, DSA, MOMA, …
  - neutral layer: generic middleware components, independent from distribution models
  - protocol personalities: GIOP, SOAP, …
- canonical architecture
  - 7 fundamental services: addressing, binding, representation, protocol, transport, activation, execution
  - a central component for coordination: µBroker
- adaptability
  - selection & adaptation of components
  - use of personalities that are adapted to applications and to needed protocols
- reliability
  - µBroker modeled using Petri nets
New properties

- **node location**
  - `GAIA::location : aadlstring`
  - applies to processors
  - interpreted as an IP address
  - actual interpretation depends on the target network/generator

- **port**
  - `GAIA::port_number : aadlinteger`
  - applies to processes (i.e. nodes)
  - interpreted as an IP port number
  - may be computed automatically
Use of a complete middleware

• AADL runtime = middleware
  • AADL thread
    • thread provided by the middleware
    • or system thread if the middleware does not provide this facility
  • communications managed by the middleware
• code generator in two parts
  • generation of applicative elements
    • from AADL subprograms & data components
  • generation of a software layer to adapt the middleware to the application
    • from AADL threads
    • translation of the different distribution models, depending on the middleware capabilities
• applicative components run by the runtime
  • the middleware controls the application
  • the middleware is configured, according to the information provided by the AADL description
  • architecture analysis
    • if we know the characteristics of the middleware
• ex.: Gaia/PolyORB
The adaptation layer

- AADL port identified by incoming request names
- Data port requests
  - Provide data that are stored into the buffer (no queue)
  - Do not trigger the controller
- Event data port requests
  - Provide data that are stored into the buffer (queue)
  - May trigger the controller
- Periodic threads
  - The runtime triggers the controller
- The controller
  - Gets the required data from the buffer
  - Calls the application (according to the call sequence of the AADL thread)
  - Get the out parameters back
Use of a minimal runtime

- transformation of the AADL description
  - expansion of AADL threads pour to insert the middleware services
  - generation of the associated code, or link with existing code
- minimal and verifiable runtime
  - execution control by the \( \mu \text{Broker} \)
    - modeled using Petri nets
  - communications using a socket library
    - AADL API
    - provided by the OS, or modeled in AADL
- the whole application is almost completely described in AADL
  - facilitates the analysis of the application
  - several programming languages can be used
  - fully generic middleware
- ex.: Gaia/\( \mu \text{Broker} \)
  - AADL runtime = PolyORB \( \mu \text{Broker} \)
    - implements the scheduling protocol
  - middleware components are parts of the application
Middleware components in AADL

- no more distribution models
  - we are at a low lever of modeling
- services modeled as subprograms
  - call sequences
  - algorithms provided as source code
    - may be generated from formal method representations
Gaia architecture

- architecture expander used to generate the PolyORB instance in AADL
  - code generation for the AADL components
- runtime generator
  - complete PolyORB instance
  - or minimal runtime
- translator for the application components
Conclusion

• clear separation of node parts:
  • runtime : AADL threads
  • application : subprograms & data
• the AADL runtime controls the application elements
  • help ensure the consistency between the AADL description and the actual application

• 2 possible approaches for the runtime:
  • a “high-level” runtime
    • provides all distribution models
    • distribution middleware + thread library
  • a “low-level” runtime
    • communication mechanisms described in AADL
    • thread library + AADL API for sockets

• http://ocarina.enst.fr
• https://libre2.adacore.com/polyorb