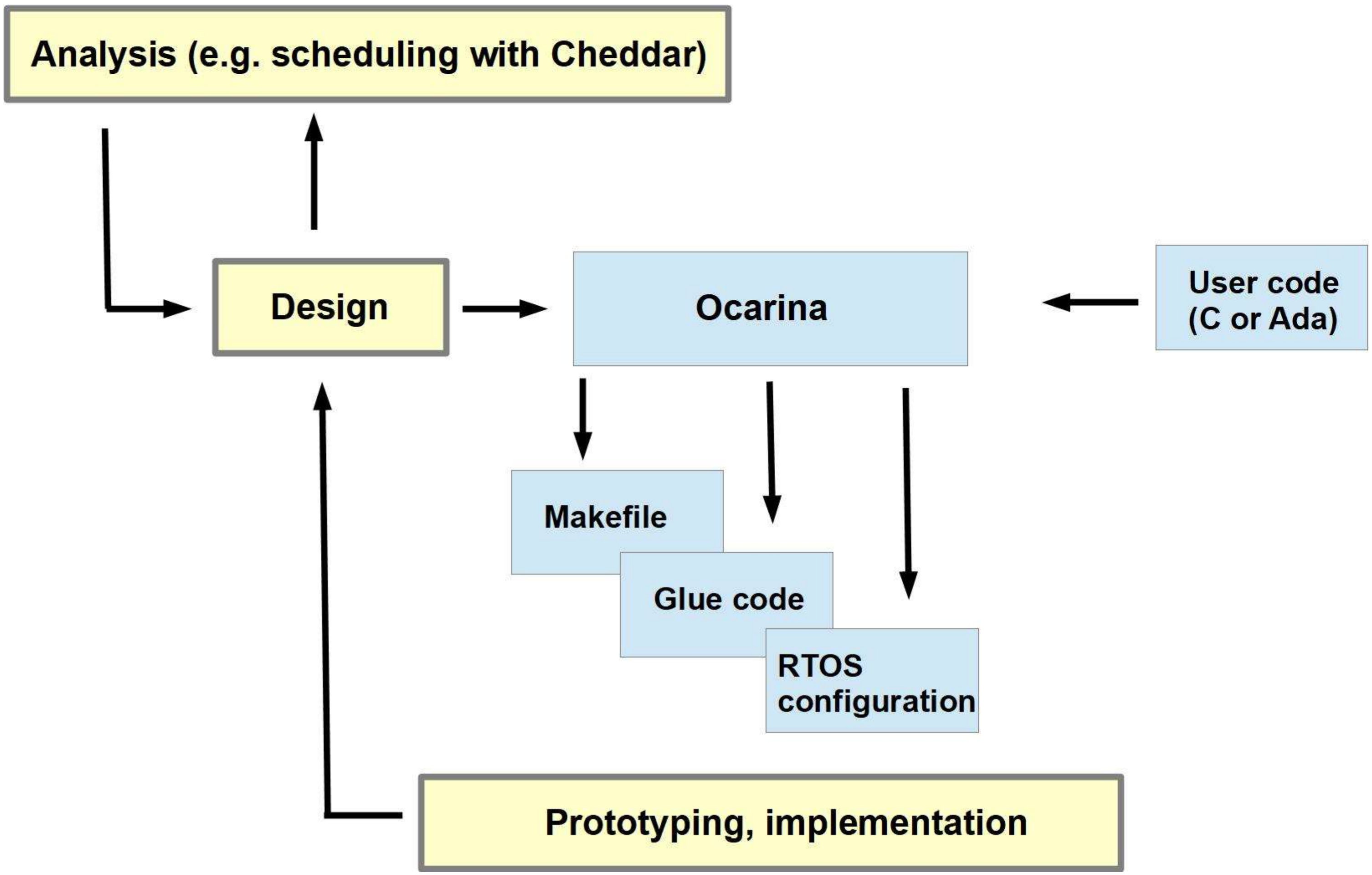
AADL code generation

Code generation

- Users write functional part, i.e. services provided by the application
  - C or Ada
- Tools use architecture specification to deploy user code, with code generation
- What kind of generated code do we need
  - Relationships with the programming language
  - Relationships with the operating system
    - Service calls
    - Configuration
  - Binary production files, e.g. makefile, gpr files
  - Running material, e.g. QEMU scripts

Process example



Data modeling annex

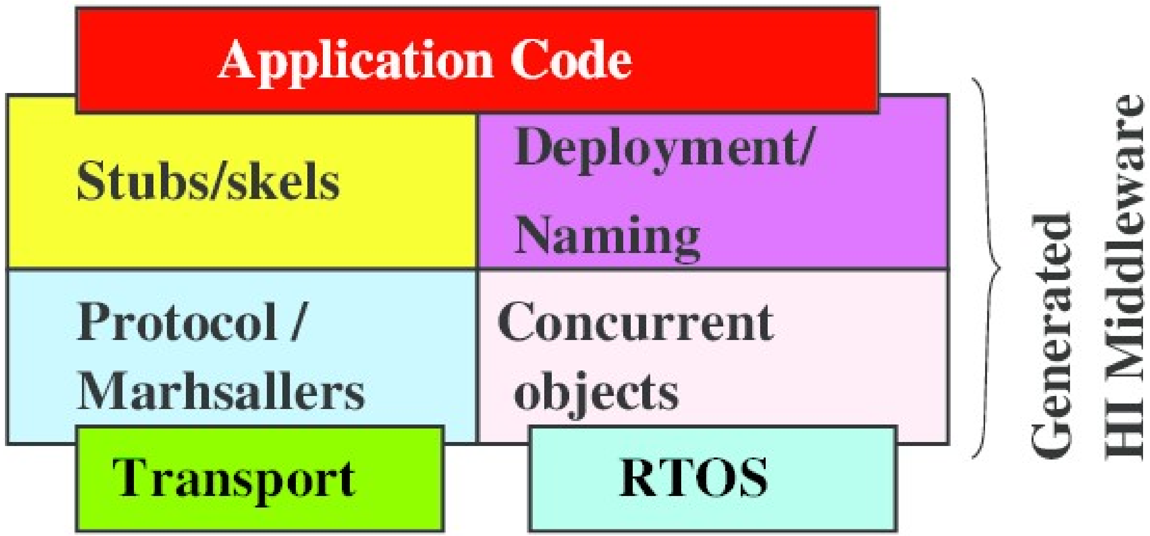
```
property set Data_Model is
   Data_Representation : enumeration
     (Array, Boolean, Bounded_Array, Character, Enum,
      Float, Fixed, Integer, String, Struct, Union)
      applies to ( data );
  --  The Data_Representation property may be used to specify the
  --  representation of simple or composite data types within the
  --  programming language source code.
  --  Note: An implementation is allowed to support only a subset
  --  of these structures.

   Enumerators : list of aadlstring applies to ( data );
  --  The Enumerators provides the list of enumeration litterals
  --  attached to an enumeration data component.

   Initial_Value : list of aadlstring applies to ( data,
       port, parameter );
  --  Initial_Value specifies a list of initial values for a data
  --  component or port in string form. For a subprogram
  --  parameter, it defines a default value.

   Integer_Range : range of aadlinteger applies to ( data,
            port, parameter );
  --  Integer_Range specifies a range of integer values that apply to
  --  the data component.  This property is used to represent integer
  --  range constraints on data that is of some integer type.

   Real_Range: range of aadlreal applies to ( data, port,
            parameter );
  --  Real_Range specifies a range of real values that apply to the
  --  data component.  This property is used to represent real range
  --  constraints on data that is of some real type.

   Representation : list of aadlstring applies to ( data );
  -- Representation specifies the actual representation of
  -- enumerators value.

   ...
end Data_Model;
```

Ocarina in few words

- AADL parser and code generator
- Generate files for analysis tools :
  - Cheddar (scheduling analysis), MAST (scheduling analysis), BoundT (WCET analysis), Petri nets, REAL (constraint analysis), Alloy
- Generate source code for execution platforms based on PolyORB HI C and Ada
  - Linux, Xenomai
  - RTOS: RTEMS, VxWorks, FreeRTOS
  - ARINC 653: Pok, Deos, VxWorks653, Xtratum

Polyorb High Integrity

- Middleware, making user code portable
- Configurable
- Part of the code is generated



Periodic tasks, user code

- Focus only on what matters and not on scheduling/syncronization/communication technical part

```
#include <stdio.h>
#include <po_hi_time.h>
void tic_spg (void) {
        printf ("[%d] Tic\n", milliseconds_since_epoch());
        fflush (stdout);
}
void tac_spg (void)  {
        printf ("[%d] Tac\n", milliseconds_since_epoch());
        fflush (stdout);
}
```

Periodic tasks, model

- Properties driving
  - Parsing of AADL models
  - Generating of glue code
  - Generating of binary production file

```
system root
properties
    Ocarina_Config::Timeout_Property        => 4000ms;
    Ocarina_Config::Referencial_Files       => ("node_a",
          "node_a.ref");
    Ocarina_Config::AADL_Files              => ("tictac.aadl");
    Ocarina_Config::Generator               =>  polyorb_hi_c;
    …
```

Periodic tasks, model

```
subprogram tic_spg
properties
    source_language => C;
    source_name     => "tic_spg";
    source_text     => ("tictac.c");
end tic_spg;

process node_a …
thread tic …
processor implementation cpu …
  Deployment::Execution_Platform => native;
  Scheduling_Protocol => …
system implementation …
  Actual_Processor_Binding =>
       (reference (cpu)) applies to node_a;
```

Periodic tasks, generated code

- **Configuration and glue code**
  - Main entry point (main.c)
  - AADL thread source code (activity.h and activity.c)
  - Configuration (deployment.h and deployment.c)
  - AADL data types (types.h and types.c)
  - AADL subprogram source code (subprograms.h and subprograms.c)
  - AADL ports (request.h and request.c)
  - Marshalling services (distributed systems)

Periodic tasks, generated code

- **deployment.c and deployment.h**

```
/**************************************************/
/*  This file was automatically generated by Ocarina */
/*  Do NOT hand-modify this file, as your            */
/*  changes will be lost when you re-run Ocarina     */
/**************************************************/
#define __PO_HI_MY_NODE node_a_k

#define __PO_HI_NB_TASKS 2
#define __PO_HI_TASKS_STACK 0
#define __PO_HI_NB_PROTECTED 0
#define __PO_HI_NB_NODES 1
#define __PO_HI_NB_ENTITIES 2
#define __PO_HI_NB_PORTS 0
#define __PO_HI_NB_DEVICES 0
#define __PO_HI_NB_BUSES 0
#define __PO_HI_NB_PROTOCOLS 0
```

Periodic tasks, generated code

- **activity.c and activity.h**

```
void* tic_job (void){
        /* Waiting for other tasks initialization */
        __po_hi_wait_initialization ();
        __po_hi_compute_next_period (node_a_tic_k);

        /* Waiting for the first dispatch instant */
        __po_hi_wait_for_next_period (node_a_tic_k);
        while (1)  {    /*  Call implementation*/
                tictac__tic_spg ();
                __po_hi_wait_for_next_period (node_a_tic_k);
        }
}
```

Periodic tasks, generated code

- **main.c**

```
PO_HI_MAIN_TYPE __PO_HI_MAIN_NAME (void){
  __po_hi_time_t period;
  __po_hi_initialize ();

  __po_hi_milliseconds (&(period), 100);
  __po_hi_create_periodic_task (node_a_tic_k, &(period), 1,
                 0, 0, tic_job);
  __po_hi_milliseconds (&(period), 50);
  __po_hi_create_periodic_task (node_a_tac_k, &(period), 2,
                 0, 0, tac_job);

  __po_hi_wait_initialization ();
  __po_hi_wait_for_tasks ();
  return (__PO_HI_MAIN_RETURN);
}
```

Periodic tasks, Ada support

- **Specific subprogram properties**

```
subprogram tic_spg
properties
    source_language => Ada95;
    -- “package_name.procedure_name”
    source_name     => "tictac.tic_spg";
end tic_spg;

subprogram tac_spg
properties
    source_language => Ada95;
    source_name     => "tictac.tac_spg";
end tac_spg;
```

Periodic tasks, Ada support

- **User code**

```
with PolyORB_HI.Output;
use PolyORB_HI.Output;

package body tictac is
  procedure tic_spg is
  begin
    put_line("Tic is running");
  end tic_spg;
  procedure tac_spg is
    begin
      put_line("Tac is running");
  end tac_spg;
end tictac;
```

Shared data, model

```
data Counter
end Counter;
data implementation Counter.Impl
properties
  Data_Model::Data_Representation => Integer;
  Priority => 250;
  Concurrency_Control_Protocol => Priority_Ceiling;
end Counter.Impl;

subprogram Read_Counter_Spg
features
  this : requires data access Counter.Impl;
properties
  source_language => C;
  source_name     => "read_counter_spg";
  source_text     => ("tictac.c");
end Read_Counter;
```

Shared data, user code

- Application is a set of subprograms. Can we call write_counter_spg in tic_spg?

```
#include <stdio.h>
…

void tic_spg (void) …
void tac_spg (void) …

void read_counter_spg (int* value){
        printf ("Read counter: %d\n", *value);
}
void write_counter_spg (int* value){
        int v = *value;  v++;  *value = v;
        printf ("Write counter: %d\n", *value);
}
```

Shared data, generated code

- **main.c**

```
#include <activity.h>
#include <po_hi_task.h>
…
tictac__counter_impl counter;
```

- **deployment.h and deployment.c**

```
#include <po_hi_protected.h>
#define __PO_HI_NB_TASKS 2
#define __PO_HI_NB_PROTECTED 1
__po_hi_protected_protocol_t
        __po_hi_protected_configuration[__PO_HI_NB_PROTECTED] =
        {__PO_HI_PROTECTED_PCP};
__po_hi_uint8_t __po_hi_protected_priorities[__PO_HI_NB_PROTECTED] =
        {250};
```

Architecture exploration

- **Designing an architecture model and having tools to produce simulation and prototypes, allow design space exploration**
  - Ranging deployment, i.e. from local versus distributed
  - Ranging synchronization/communication tools, shared data versus (event) data port
  - Ranging priorities of entities
  - Ranging scheduling policies, i.e. preemptive versus non preemptive
  - …

Software design to deploy

```
data implementation Counter.Impl
properties
  Data_Model::Data_Representation => Integer;
end Counter.Impl;

thread P
features
  Data_Source : out event data port Counter.Impl;
end P;
thread Q
features
  Data_Sink : in event data port Counter.Impl;
end Q;
```

Software design to deploy

```
thread implementation P.Impl
calls Mycalls: {P_Spg : subprogram Do_Ping_Spg;};
connections
  parameter P_Spg.Data_Source -> Data_Source;
end P.Impl;

thread implementation Q.Impl
calls Mycalls: {Q_Spg : subprogram Ping_Spg;};
connections
  parameter Data_Sink -> Q_Spg.Data_Sink;
end Q.Impl;
```

Software design to deploy

```
#include <stdio.h>

int p=0;

void do_ping_spg (int *v) {
  printf ("*** SENDING PING *** %d\n", p);
  *v=p;  p++;
  fflush (stdout);
}

void ping_spg (int i){
  printf ("*** RECEIVING PING *** %d\n" ,i);
  fflush (stdout);
}
```

Explore local deployment

```
processor cpu …
process implementation A.Impl
subcomponents
  Pinger  : thread P.Impl;
  Ping_Me : thread Q.Impl;
connections
  port Pinger.Data_Source -> Ping_Me.Data_Sink;
end A.Impl;

system implementation local.impl
subcomponents
  Node_A : process   A.Impl;
  A_cpu  : processor cpu;
properties
  actual_processor_binding => (reference (A_cpu))
       applies to Node_A;
```

Explore distributed deployment

```
process A
features  Out_Port : out
      event data port  Counter.Impl;
end A;

process B
features  In_Port  : in
      event data port Counter.Impl;
end B;
```

Explore distributed deployment

```
process implementation A.Impl
subcomponents
  Pinger : thread P.Impl;
connections
  port Pinger.Data_Source -> Out_Port;
…

process implementation B.Impl
subcomponents
   Ping_Me : thread Q.Impl;
connections
   port In_Port -> Ping_Me.Data_Sink;
…
```

Explore distributed deployment

```
system implementation distributed.Impl
subcomponents
  Node_A : process A.Impl;   Node_B : process B.Impl;
  Dev_A  : device …          Dev_B  : device …
  Cpu_A  : processor cpu;    Cpu_B  : processor cpu;
  A_bus  : bus ocarina_buses::ip.i;
connections
  bus  access A_bus     -> Dev_A.link;
  bus  access A_bus     -> Dev_B.link;
  port Node_A.Out_Port -> Node_B.In_Port …
properties
  actual_processor_binding => reference (Cpu_A) applies to Node_A;
  actual_processor_binding => reference (Cpu_B) applies to Node_B;
  actual_processor_binding => reference (Cpu_A) applies to Dev_A;
  actual_processor_binding => reference (Cpu_B) applies to Dev_B;
```