UML AADL 2009 Workshop
Potsdam - Germany

UML Modeling and Formal Verification of control/data driven Embedded Systems

Presented by: Fateh Boutekkouk

University Larbi Ben M'hedi of Oum El Bouaghi – Algeria-

Outlines (1)

- **Introduction and Motivation**
- **Introduction to Rewriting logic & Maude Language**
- **Our Approach**
  - ✓ Application modeling
  - ✓ Architecture modeling
  - ✓ Mapping modeling
- **Transformation of UML models to Maude**

Outlines (2)

- **Tasks behaviors specification**
- **Properties specification and verification**
- **Example**
- **Conclusion and perspectives**

Introduction and Motivation (1)

- The ever increasing complexity in both applications and integration density in semi-conductor technology, and strict time to market have pushed Embedded Systems specialists to:
  1. raise the level of abstraction
  2. borrow some technologies and ideas from software engineering such as Object technology and formal techniques
- UML is the de facto standard for visual object modeling
- UML can be tailored to different application domains by the definition of profiles
- Since UML does not dictate any particular development process to be used, it is on designers to define a design flow. We think that the Y-chart approach is the most appropriate.

Introduction and Motivation (2)

- UML lacks a formal support for early verification and validation
- Transformation of UML models to a language with a well defined semantics
- We use this formal language to verify the correctness of the system against some undesirable properties and eventually perform high level estimations on system performances (eg. Power consumption)

Introduction to Rewriting Logic and Maude Language (1)

- The rewriting logic (RL) was introduced by Meseguer.
- In RL, the logic formulas are called rewriting rules. They have the following form: **R: [t] -> [t'] if C**. Rule R indicates that term t is transformed into t' if a certain condition C if verified.
- Term represents a partial state of a global state S of the described system.
- The modification of the global state S of the system to another state S' is realized by the parallel rewriting of one or more terms that express the partial states.

Introduction to Rewriting Logic and Maude Language (2)

- Maude is a specification language based on the rewriting logic.
- Two specifications level are defined: the system specification & properties specification
- The system specification level is provided by the rewrite theory.
- three types of modules are defined in Maude.
- Functional modules allow defining data types and their functions through equations theory.
- System modules define the dynamic behavior of a system by introducing rewriting rules.
- Object-Oriented modules

Introduction to Rewriting Logic and Maude Language (3)

```
mod BANK-ACCOUNT is
protecting INT .
including CONFIGURATION .
op Account : -> Cid.
op bal :_ : Int -> Attribute .
ops credit debit : Oid Nat -> Msg .
var A : Oid . vars M N :Int
rl [credit]: < A : Account | bal : N > credit(A, M) => < A : Account | bal:N + M >
crl [debit] : < A : Account | bal : N > debit(A, M) => < A : Account | bal : N - M >
     If  N >= M .
endm
```

The BANK-ACCOUNT module in system module form.

Introduction to Rewriting Logic and Maude Language (4)

- The property specification level defines the system properties to be verified.
- The system is described using a system module.
- The Model-checking supported by Maude's platform essentially uses the LTL logic for its simplicity and the defined decision procedures it offers.
- The user can call the *modelCheck* function while specifying a given initial state and a formula.
- Maude model-Checker verifies if this formula is valid in this state or the set of all reachable states form the initial state.
- If the formula is not valid, a counter example (*counterexample*) is displayed.

Our approach (1)

➢ Application Modeling

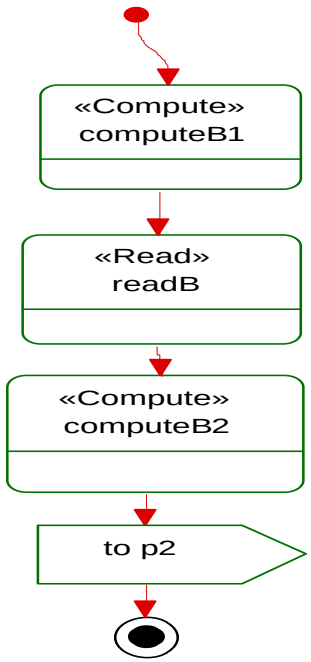
- Separation between data driven and control driven tasks
- Separation between computing and communication
- hierarchy
- Data abstraction
- Definition of a set of stereotypes:
- Module
- Dtask
- Ctask
- CChannel
- DChannel

Our approach (2)

- Data Driven tasks
- we use UML2.0 activity diagrams with Coarse Grained Actions (CGAs)
- Each CGA belongs to one of the three generic types: Computation Actions (CAs), Read Actions (RAs), or Write Actions (WAs).
- we define a new stereotype called "*Compute" with* ctwo tagged values : the number of elementary instructions inside a computation, and the type of the elementary operation (i.e. integer or float).

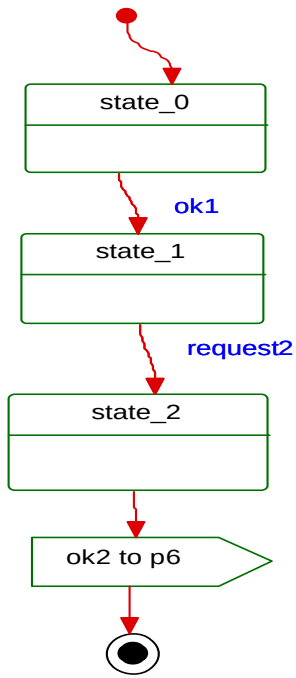
Our approach (3)

- Data Driven tasks
- We use UML activity diagrams with croase grained actions (CGAs)



Our approach (4)

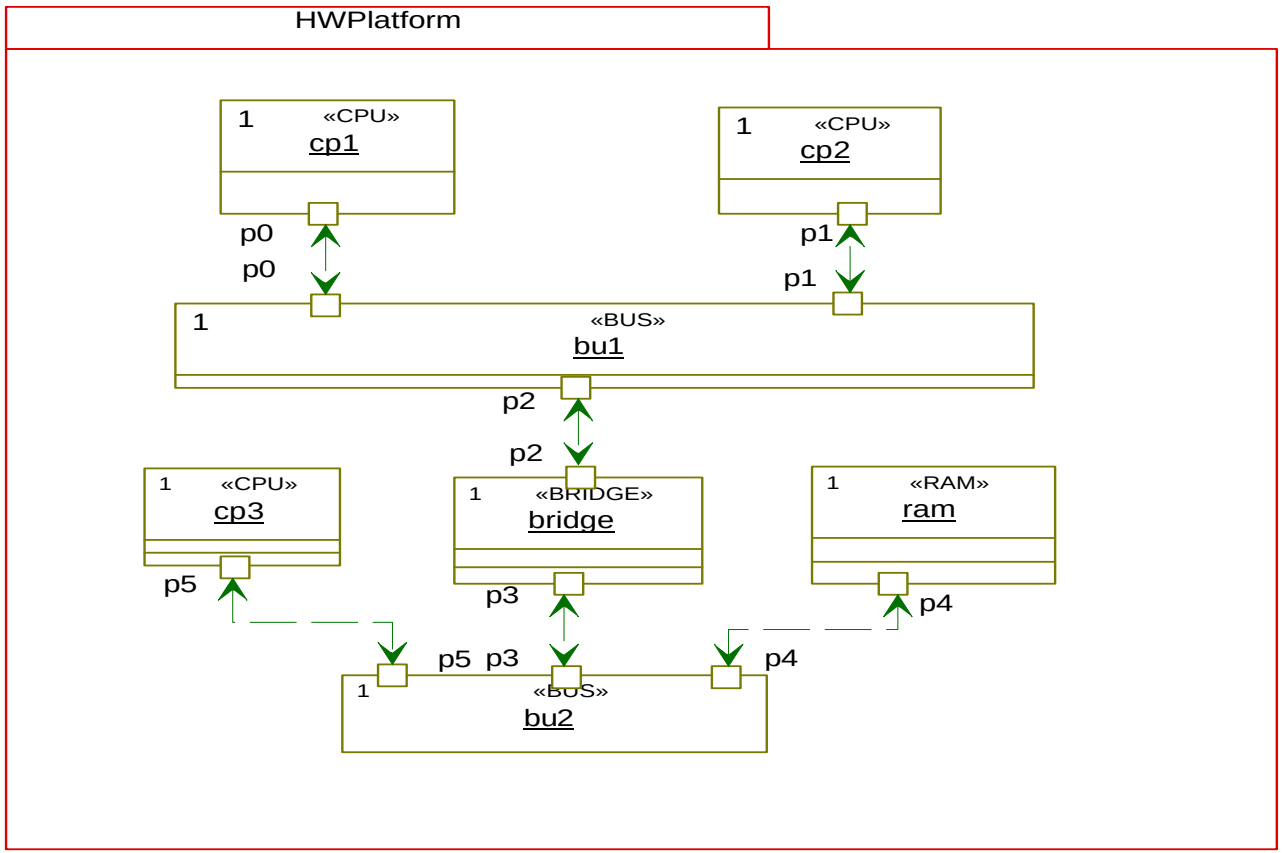
- Control Driven tasks
- We use UML stateCharts with zero time



Our approach (5)

- Architecture Modeling
- A set of stereotypes are defined: CPU; BUS; BRIDGE; RAM
- CPU
- Task switching time
- the time to go to the “idle” state,
- the number of cycles for an elementary operation,
- the average amount of power consumed per cycle in the running mode,
- The average amount of power consumed per cycle in the idle mode,
- The scheduling algorithm.

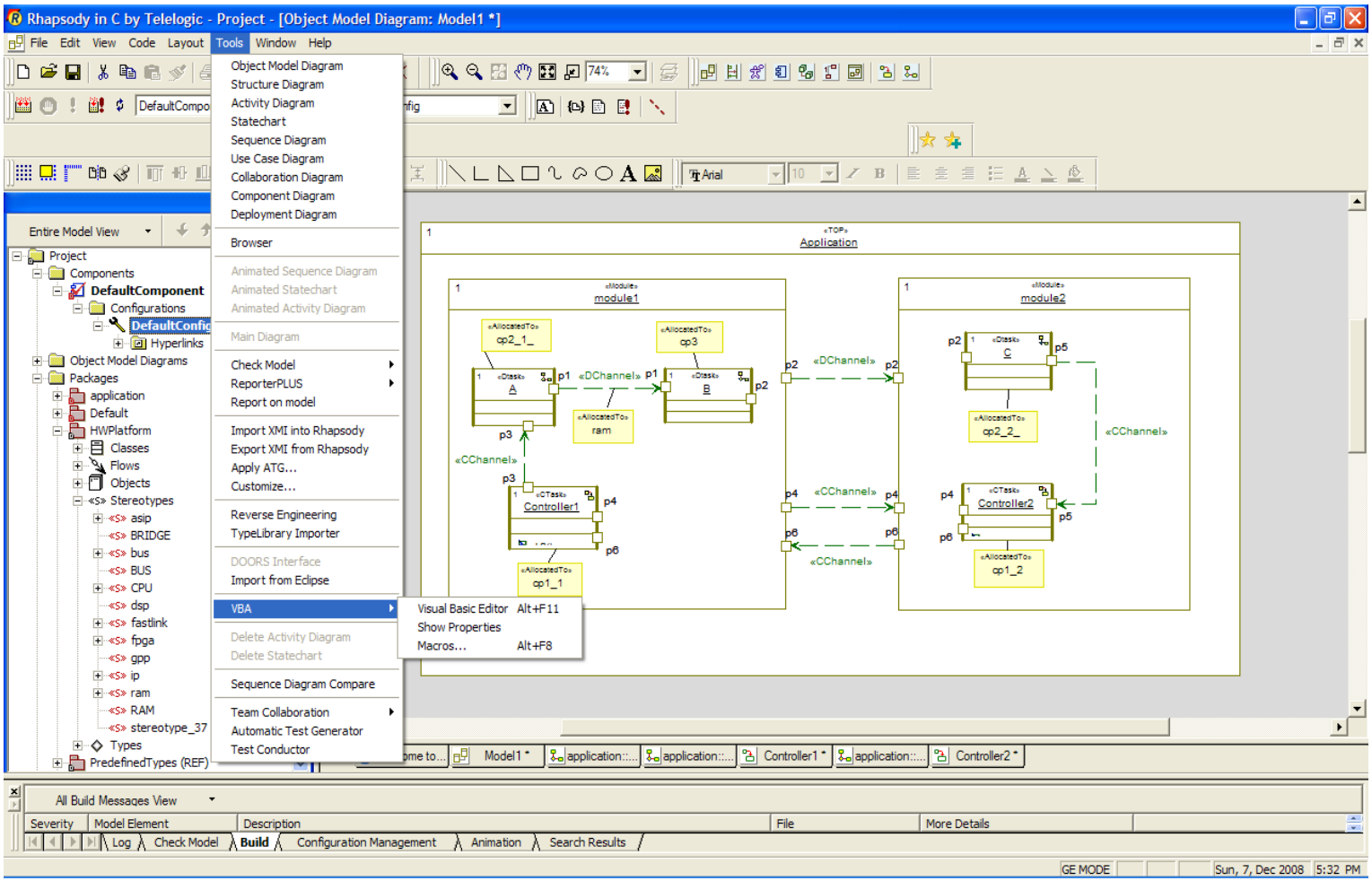
Our approach (6)



Our approach (7)

- Mapping modeling
- We define a new stereotype called *"AllocatedTo"*.
- This stereotype is applied on the UML constraint and it has two stereotypes. The first one specifies the name of the hardware component to which logical component should be allocated. The second one designates a number that determines the execution order of the task (the transfer).

Our approach (8)



Modeling using Rhapsody

Passage from UML to Maude (1)

- Transformation of UML models to Maude

UML	Maude
Composite class (Module)	System Module
Class (Task)	Class
SysML Flow (Channel)	Class
Tagged value	Attribute
CGA	Attribute
FSM state	Attribute
Activity diagram/FSM Transition	Rewriting rule

Passage from UML to Maude (2)

- < A: Dtask | hwname: cpu, state: sta, action: act >
- < B: Ctask | hwname: cpu, state: sta, FSMS: fsms >
- < chd : Dchannel | hwname : hw, source : A, target : B, available : x >
- < chc : Cchannel | hwname : hw, source : A, target : B, size : x >
- *< cpu : CPU | LinkTo : bus, ContextSwitch : cont, GoIdle : idl, Iop : iop, Fop : fop, Power : pw, PowIdle : pwd, TPower : tp >*
- *<bus : BUS | Speed : sp, Power : pb, TPower : tpb, free : true >*

Tasks Behaviors specification (1)

- *rl [start] : ***1*

start(A)
< A : Dtask | hwname : cpu, state : ready, > =>
< A : Dtask | hwname : cpu1, state : run, action : readC, token : 5 > .

Tasks Behaviors specification (2)

- *crl [readCwait] : ***2*

< A : Dtask | hwname : cpu, state : run, action : readC, token : n > < cpu : CPU | LinkTo : bus, ContextSwitch: cont, GoIdle : idl, Iop : iop, Fop : fop, Power : pw, PowIdle : pwd, TPower : tp > < ch1 : Dchannel | hwname : bus1, source : A, target : B, available : x > < C : Dtask | hwname : cpu, state : s >

=>

*< A : task | hwname : cpu, state : wait, action : readC, token : n – x > < cpu : CPU | LinkTo : bus, ContextSwitch : cont, GoIdle : idl,Iop : iop, Fop : fop, Power : pw, PowIdle : pwd, TPower : tp + (float(cont) * pw) > < ch1 : Dchannel | hwname : bus1, source : A, target : B, available : 0 > < C : Dtask | hwname : cpu, state : s > wakeup(C) if (x < n) and (s == ready) .*

Tasks Behaviors specification (3)

- *crl [EV5occurence] : ***3*

< A : Dtask | hwname : cpu, state : run, action : waitEV5 >

< ch : Cchannel | hwname : bus1, source : A, target : B, size : sz >

=>

< A : Dtask | hwname : cpu, state : run, action : computeDCT, Nombre : 1000, Type : integer > < ch : Cchannel | hwname : bus1, source : A, target : B, size : sz – 1 > if sz > 0 .

Tasks Behaviors specification (4)

- *rl [computeDCT] : ***4*

< A : Dtask | hwname : cpu, state : run, action : computeDCT, Nombre : 1000, Type : integer >

< cpu : CPU | LinkTo : bus ,ContextSwitch : cont, GoIdle : idl, Iop : iop, Fop : fop, Power : pw, PowIdle : pwd, TPower : tp >

=>

< A : Dtask | hwname : cpu, state : run, action : writeA, token : 5 >

*< cpu : CPU | LinkTo : bus, ContextSwitch : cont, GoIdle : idl, Iop : iop, Fop : fop, Power : pw, PowIdle : pwd, TPower : tp + (float(1000 * iop)* pw) > .*

Properties specification and verification (1)

- At this level of abstraction, we can verify some undesirable or/and desirable properties.
- Using the Maude command ***"search in application: initial =>! X:conf such that TaskEnd(X:conf) == true."***, we can easily verify whether all tasks reach the idle state (that means there is no deadlock).
- ***TaskEnd*** is a function defined as:

```
sort conf .
subsort conf < Configuration .
op sta : conf -> states .
op TaskEnd : conf -> Bool .
eq sta (< T1 : Dtask | hwname : cpu, state : st >) = st .
eq sta (< T2 : Ctask | hwname : cpu, state : st >) = st .
eq TaskEnd (T) = if sta(T) == idle then true else false fi .
```

Properties specification and verification (2)

- We can verify whether the bus is always busy which is a non-desirable property.
- For this reason we use the command ***"search in application : initial =>! X:conf such that BusBusy(X:conf) == true ."***
- ***BusBusy*** is a function defined as:

op BusBusy : conf -> Bool .

eq BusBusy (< bus : BUS | Speed : sp, Power : pb, TPower : tpb, free : bool >) = if bool == false then true else false fi .

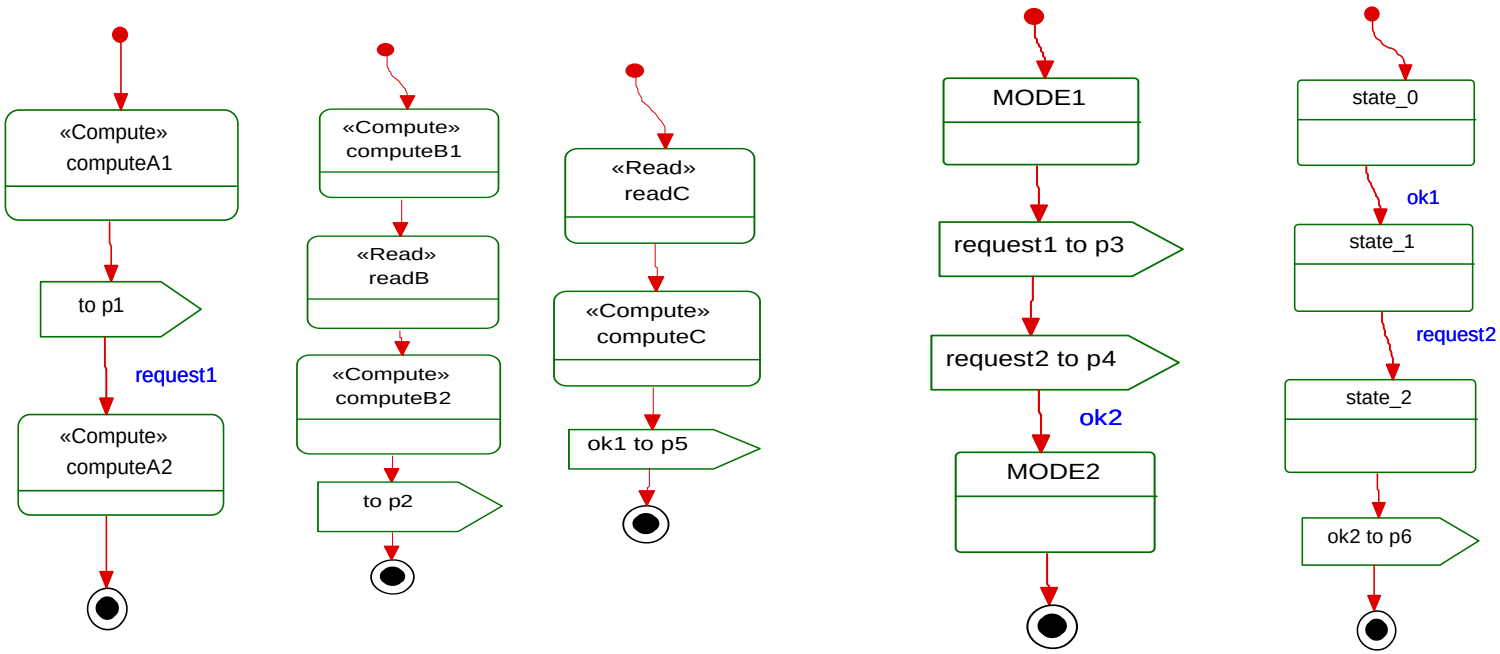
Properties specification and verification (3)

- Another important property we can verify is the fact that the amount of data tokens buffered in data channels FIFO does not exceed a certain threshold in every state of the system.
- Using the command: ***"red modelCheck(initial, []FIFOsize(initial)) ."***, we can easily verify the FIFOsize property
- ***FIFOsize*** is defined as:

```
var cf : configuration .
op FIFOsize : Configuration -> Prop .
ops ch A B bus : -> Oid .
vars x TS : Nat .
ceq < ch : Dchannel | hwname : bus, source : A, target : B,
    available : x, threshold : TS > cf |= FIFOsize (< ch : Dchannel |
    hwname : bus, source : A, target : B, available : x, threshold :
    TS > cf) = true  if x < TS .
```

Example (1)

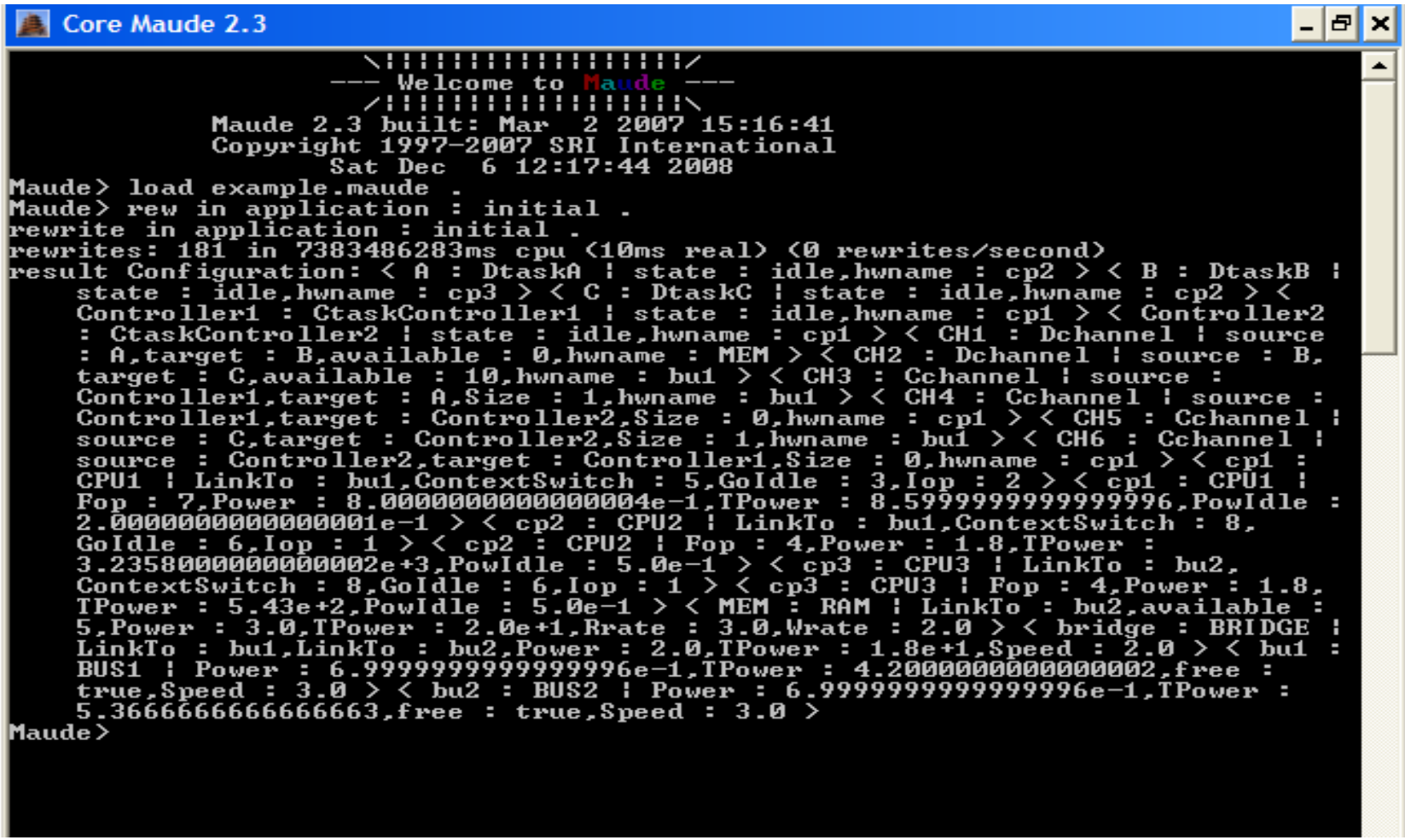
- Three data driven tasks: A, B, and C
- Two controllers: Controller1 and Controller2



Example (2)

- Task *A* performs a CGA *computeA1* including 300 integer operations, and writes 10 tokens of data over *ch1*. Then, it waits for *request1* event from *Controller1* (via *ch3*) to perform a second CGA *computeA2* including 120 float operations before it terminates.
- Task *B* performs a CGA *computeB1* including 100 integer operations and attempts to read 5 data tokens from *ch1*, then it performs *computeB2* including 50 float operations, and writes 8 data tokens over *ch2* before its termination.
- Task *C* attempts to read 5 data tokens from *ch2*, performs a CGA *computeC* including 1000 integer operations, then it sends an event *ok1* to *Controller2* over *ch5*, and terminates.
- Before execution starts, we assume that the number of available tokens for *ch1* is equal to 0, and 7 for *ch2*

Example (3): Rewriting Result



Conclusion and perspectives

- Use of UML as a front-end for Data/Control Driven Embedded Systems Modeling.
- Transformation of UML models to Maude language.
- The passage from UML to Maude is done into Rhapsody by mean of its VB interpreter.
- Formal verification of some properties and high level Performances estimation using Maude rewriting engine.

As a perspective:

- Enrich our model by adding more realistic information concerning time and power consumption.
- Use of ATL for expression of transformation rules
- Discover other properties for formal verification.

UML AADL 2009 Workshop
Potsdam - Germany

Thank you