COntinuuM, a CO-modelling Methodology for the Integration of Real-time Architecture Models

I Perseil, Laurent Pautet

COntinuuM, a CO-modelling Methodology for the Integration of Real-time Architecture Models

I. Perseil, L. Pautet

TELECOM Paristech – LTCI-UMR 5141 CNRS
46, rue Barrault, F-75634 Paris cedex 13, France
isabelle.perseil@enst.fr, laurent.pautet@enst.fr

Abstract: The design of Distributed Real-time Embedded (DRE) architecture models for complex and critical systems with safety, liveness, timeliness, dependability concerns, forces the use of formal languages.

Because of the high level of criticity, proof techniques are required instead of model-checking with limitations relatively to the state space explosion problems.
Proofs of these non-functional properties can only be established on the basis of formal languages with high verification capabilities (theorem provers).

Therefore, we have concentrated our efforts on the development of a methodology that would better integrate formal aspects into the design of DRE architectures, which is usually based upon the use of (semi-formal) Architecture Design Languages (ADLs). This methodology has both to support the traceability of non-functional property proofs (from the requirements to the deployment of a DRE system) and the integration of formal and non formal modelling languages.

The approach is bottom-up when the method states that each realization artifact, even hidden, has to be detected from the capture requirement stage (each possible realization artifact has to be identified during a prototype coding stage)
As a consequence, language translations are not based on the MDA process that supposes some projections. These projections would be responsible for the gap between abstractions used to understand and describe the problem and those used for implementation.
To bridge those gaps is the major aim of the methodology, so we called it “Continuum” as it would help to restore the development process continuity.

The new aspects of this methodology (and its difficulties) are essentially the introduction of low level concepts (needed for the implementation stages) into the modeling language structures, usually more generic.

The methodology application is the development of an algorithmic language translator that enable the generation of a safe code.

Keywords: Co-modeling, ADLs, Real-time Systems, Distributed and concurrent algorithms, Proof-based system engineering

1. Introduction

Architectural configurations of critical real-time systems have to be formally verified. Non-functional properties are often specified through algorithms that mainly involve process units, but these algorithms are not described so that we can easily trace both their impact and accuracy. Many of the architecture design languages are providing features concerning the dual possibility of describing software and hardware components, as well as the implementation process and non-functional property specifications. However they do not provide enough formal capabilities to describe component behavior. This is the reason while component behavior is often described separately.

Although ADLs are more focused on programming in the large, the behavior of a component is correlated to the behavior of its subcomponents. Consequently, we have chosen to build bridges between a strict and static design process that only focuses on the topology, and a dynamic process that raises all the design levels to bring the parameters of an optimal configuration to the upper levels, particularly, when we choose to implement an algorithm that involves atomic components (like threads).

Due to our new co-modeling methodology, we may follow each step of software design process without any discontinuity.
Our approach is based on the introduction of algorithmic blocks at the highest level of the design. These blocks are instanced and optimized through an iterative process that allows feedback from successive implementations.
In order to build a complete design of component behavior, we introduce methodical guidelines that

ease the work of designers. Some elements of the design will be hidden in order to offer a more conceptual view, but from step to step, it will be easier (in terms of compliance) to show implementation details, than to generate them through a classical MDA approach.

To support our approach, we have fully integrated an algorithm language, +CAL, in the architecture design process, so that we can gather a code issued from an ADL (AADL) specification and a code issued from the +CAL specification. Therefore, we will present the +CAL/Ada translator we have realized and its integration in our Ocarina toolset that has been previously designed for AADL translations to Ada. The produced Ada code will be validate, once the TLA+ specifications, automatically generated from the +CAL specifications, are checked.

Introducing every concept in the earliest phases of design that might be taken in account by the final code, we save intermediate translations and model transformations which are often hard to validate.
So at the same time, we dramatically reduce the software costs, improve the reuse and reinforce the necessary proof based system engineering for the critical real-time system we design.

2. Background

2.1 Context

Methods have been supplanted by standards.

In the past, Methods dedicated to real-time systems design, like SART, for instance, had a very large scope and were applied in many fields. Systems were less complex and it was easier to adapt a generic method.

Then came a strong wish to standardize best practices, and to start with the modelling language.

After a while, the standardization process reached the patterns and then specific metamodels, e.g. UML profiles.

2.2 State-of-the-art (Related works)

Analysis and design of real-time architectures are realized without any standardized process or methods. Some well-known processes do exist but are not uniformly applied:

- Avionic standards as DO-178B, ARP4754 et proprietary guidelines
- MDE / MDA, a UML Profile for MARTE, xUML, a UML Profile for AADL
- The set MetaH + ControlH
- The methodology Proof Based System Engineering (PBSE)

The PBSE methodology, which is the most advanced among the proof methodologies, describes a potential cycle with feedback, but suffers from a lack of guidelines and has no language (neither theorem prover) support, in addition the use of this methodology is not yet applied with any modeling language.

A lot of research attempts of model-checking on semi formal ADLs models (AADL with CPN) or formal ADLs models (with Wright, Acme or Rapide) linked with non formal requirements exist with some results, but are not adopted by industrials, because of the implementation complexity.

Nowadays, a lot of prototyping methodologies are co-existing, far away to be generalized or standardized, and moreover, these methodologies are generally bypassing the requirement capture stages, taking the domain understanding and its modeling for granted.

So far, there exists a gap between the abstractions used to formalize the problem domain and those for the implementation. The MDA approach proposes some model transformations techniques to bridge the gap but this is not the only way to manage it.

2.3 Issues

This is a language integration problem. Languages are defining a system at different granularity levels, from one end to the other end of the life cycle.

MDA is not the solution for embedded systems, because the physical environment has an impact on the system behavior, so the software and the run-time framework cannot be separated, as it is proposed in the MDA approach.

The integration problem is located at many levels:

- There is no continuity between the development stages: requirements, analysis, design, coding, testing and verifications, 100% of traceability is far from reality, so how to rely different levels of abstraction?
- The granularity and formalism language integration is difficult, considering a system through very different aspects: so how to rely these languages that have different levels of expressivity an from different types (functional, imperative…)

2.4 Objectives

The main objective is the development of a method that can both support specification proofs (for a given problem, the specification perfectly answer to

the problem and a proof is provided) and to prove that the implementation is correct.

The other objectives are:

- The possibility to adapt concurrent algorithms, scheduling algorithms, distributed algorithms or fault tolerant algorithms to the target architecture configuration or vice versa in order to ensure a reliable architecture configuration: a best behavior expressivity and the proof on valid states.
- The methodology will be particularly achieved to answer to the co-design issues, when dealing with an embedded system that requires to reduce to the maximum the number of components meanwhile it has to answer to hard real-time constraints.

3. Proposed approach

The DRE systems design require both high level and low level languages.
High level languages as algorithm languages (here +CAL) are most of the time used before describing the global architecture with an ADL (here the AADL a SAE standardized ADL), some kind of a low level language, very close to a programming language that will be used at the implementation stages.

Only the critical parts are requiring formal languages, so the problem to solve is really a languages integration problem.
We are striving to specify the behavior of each component in a continuous way, from the smallest granularity component (thread) behavior specification to the highest (modes configuration). Such an objective requires an integrated approach.

The main argument that leads us to expand the AADL language lies in the fact the real-time systems we are studying are using complex algorithms to specify their atomic component behavior which have a huge influence on the whole of the resulting architecture. Not including construction of algorithms in architecture design represents a high risk that they will never be totally taken into account when choosing the final architecture configuration. To ensure the requirements traceability in the analysis and design of the architecture, we consider therefore, that algorithms must appear as a significant element of the design.

Therefore, we propose to expand the AADL language in order to integrate the main algorithmic specifications that play a role in configurations. Including an algorithm language also provides an opportunity for automatic proof and clean code generation. Considering the previous target of encompassing critical system requirements, it is necessary to retrieve proof at each level of architecture design. It is not enough to claim that proofs must occur during the earliest steps of the design. The final mode configurations must be chosen using proof argumentations. This leads us to integrate a formal behavior specification language right inside the architectural specification.

The integration of an algorithm language, +CAL, into an AADL specification is under construction through an annex mechanism.

4. Application of the approach

Behavioral descriptions are associated with AADL components. Hence behaviors involving several threads cannot be directly described.
The example on figure 1 describes a situation in which describing local behavior, attached to each AADL thread, is not sufficient. Two processes are bound to different processors.

Each process is actually made of one thread and encapsulates one data component. Both data components are shared by the two threads of the architecture.

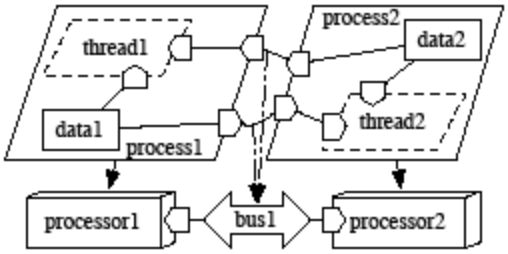

Figure 1: Data shared between two processes

Since behavioral descriptions in AADL can be associated with threads or subprograms, one can describe how the shared data should be processed by each thread. For example, it would be possible to describe that thread1 reads and writes in both data components, and that thread2 does the same, periodically. Accesses to data components are managed in each process by the AADL runtime. Therefore, we can specify a locking policy on each data component, handling the access requests sent from the AADL threads.

Descriptions of local behaviors are not sufficient to specify global behaviors. For example, we cannot ensure that the two threads access alternatively both data components, thus performing mutual exclusion on the two data components at a time. We cannot describe the management of the shared data,
and the necessary distributed lock.

We have choosen a mutex algorithm implementation that can have an influence on the resulting configuration.

This algorithm guarantees mutually exclusive access to a critical section among a number of competing processes.

```
−−algorithm bakery
variables Ex t r a c t i o n = [ k \ i n 1 . .N |−> FALSE] ,
Rank= [m \ i n 1 . .N|−> 0 ] ;
process a process \ i n 1 . .N
var iable q ;
begin
  Ex t r a c t i o n [ a process ] : = TRUE;
  Rank [ a process ] : = 1 + max(Rank [ 1 ] . . Rank [N] ) ;
  Ex t r a c t i o n [ a process ] : = FALSE;
  q :=1 ;
while q /= N+1 do
   while ( Ex t r a c t i o n [ q ] )
      do skip ;
   end while ;
  while ( ( Rank [ q ] / = 0) /\ ( ( Rank [ q ] , q ) <
        (Rank [ a process ] , a process ) ) )
  do skip ;
  end while ;
q :=q+1;
end while ;
      \_The c r i t i c a l sec t ion
      Rank [ a process ] : = 0 ;
      \_ non−c r i t i c a l sec t ion . . .
end process
end algorithm
```

Listing 1 : Lamport Bakery algorithm in +CAL

5. Integration in AADL Models

Given the following AADL description corresponding to figure 1, we will first analyze where is the most appropriate place to integrate algorithm structures.
We would rather like to insert the +CAL algorithm within the global system implementation that represents the whole architecture. This system implementation is the place where the main components (i.e. the processes, the processors, etc.) are instantiated and connected. It is also the place to describe the way data are shared.
In order to be compliant with the AADL annex behavior specification V1.5, atomic behaviors should also be attached to subprogram implementations.

From the +CAL description of the algorithm, we are able to produce source code. In order to create a complete description of the application, we have to merge the source code we generate with the description of the initial architecture (represented on figure 1).
The implementation of the algorithm in itself implies some modifications in the code executed in the AADL threads, in order to add calls to procedures such as the “entering” Ada generated procedure from the +CAL algorithm.

```
procedure enter ing ( a process : in ( proc index ) is
begin
  Ex t r a c t i o n ( a process ) := t rue ;
  Rank ( a process ) := 1 + maximum;
  Ex t r a c t i o n ( a process ) := f a l s e ;
  for q in 1 . .N loop
       loop
          delay 0 . 0 ;
          exi t when not Ex t r a c t i o n ( q ) ;
          exi t when (Rank ( q )=0)
                or (Rank ( a process)> (Rank ( q ) )
           or ( a process > q )
       end loop ;
 end loop ;
end entering ;
−−
−− Ex i t Protocol
procedure way out ( a process : in ( proc index ) is
begin
       Rank ( a process ) := 0;
end way out ;
end algo Lamport bakery ;
```

Listing 2 : Lamport Bakery algorithm in Ada

In addition, the Bakery algorithm relies on two variables, shared by all the threads. These variables have to be integrated into the architecture, as shown on figure 2.

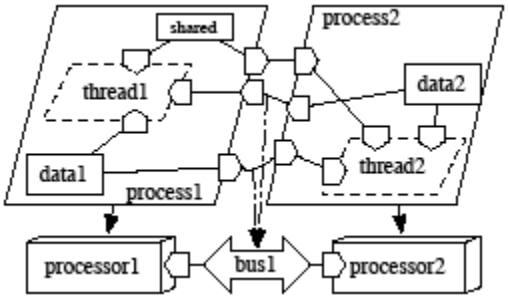

Figure 2: Architectural impact of the Bakery algorithm

The shared data is instantiated in one of the AADL processes, and accessed by all AADL threads.
The locking policy of the shared data is centralized at the level of one process, and can then be easily managed.

6. Conclusion

Architecture analysis and design is mostly performed without any standardized process or methodology. As a consequence, there is a very little traceability to handle the transition between the requirements, analysis and architecture design steps.
On the one hand, in describing the global requirements, the functional is separated from the non functional properties. What is considered the most suitable algorithm is then chosen to fit the requirements of the non-functional properties. Unfortunately, there is no going-back on the choices

we have made. In the prototype phase, it is often necessary to adapt the algorithms to the architecture configuration, and vice versa.
Our method provides a way, when choosing and updating parameters, to dynamically build an optimal configuration.
On another hand, we build architectures that follow the requirements but, make abstraction of all the behavior constraints. When the two branches of the overall development cycle meet, arises the eternal problem verifying that the architecture framework is a good foundation for the application.
Our purpose is to complete the existing gap between requirements and analysis. At the same time, the complex algorithms that we have to manage in critical systems are provided with a formal shape which allows formal proofs.

7. Glossary

AADL: (the SAE) Architecture Analysis& Design Language
ADL: Architecture Design Language
DRE: Distributed Real-time Embedded
MDA: Model Driven Architecture
+CAL : The algorithm language of Lamport
TLA: The Temporal Logic of Actions
UML: Unified Modeling Language