**How does one analyse an architecture for deadlock?**

Software architecture modeling is a relevant subject for the production of real-time systems (RTSs).

The development of architectural analysis and modeling languages in previous years has allowed

representing both structure and behavior of such systems with less consideration to implementation

details.

In this context, an architectural style is a consistent set of building elements with architecture

rules for using them to create system models. The style well-formedness rules assure a minimum

consistency level. Nevertheless, in addition to the notational or syntactic capabilities of a style, process

and guidelines are also needed to help software architects produce feasible models concerning

particular quality attributes (e.g. efficiency or safety). The Pipelines of Processes in Object-Oriented

Architectures (PPOOA) architectural style [1] has been selected because it combines UML notation

and model driven architecture (MDA) concerns, allowing for software architectural analysis and

evaluation. In addition, this style is particularly useful to explicitly represent concurrency issues.

This style has been recently selected as one of the reference MBSE methodologies by the OMG [2].

Deadlock is far from being a solved problem. Concurrency is in fact an open issue for many

research and technical work [3–7] and of high interest to the industry, especially in the real-time

embedded systems domain. Over the last three decades, different formal methods have been developed

to specify and verify system properties. In this context, model checking [8] has become a

reference discipline for such an approach. Its main goal is to build a finite model of a system and

check that relevant properties are present in it. What is remarkable about this approach is that an

exhaustive search of the state space is performed to ensure the fulfillment of some property. One of

the properties particularly checked through model checking techniques applied to concurrent systems

is deadlock absence [5, 6]. Additionally, for the classical state explosion problem (the main

drawback of formal methods), from a practitioner point of view, other relevant issues are the intrinsic

complexity of the modeling techniques and their applicability to large-scale RTSs. Industrial

applications require simple and practical approaches to be easily adopted by practitioners.

In addition to deadlock detection and prevention, the third traditional strategy is deadlock avoidance.

Under this category falls a successful mechanism that provides deadlock freedom. The group

of resource access protocols known as priority inheritance [9] has as a major objective the resolution

of priority inversion. As a collateral benefit, deadlock is also avoided if the priority ceiling protocol

(PCP) or highest locker protocol is supported by real-time operating systems. The main issue

with this mechanism is that few commercial real-time operating systems support these protocols

and their *ad hoc* implementation by the developers is complex and therefore onerous. Furthermore,

some authors have reported performance overheads derived from the utilization of these

protocols [10].

The approach presented consists in refining the cyclic complexity with additional criteria from the

structural and behavioral views of an architectural model [14]. This refinement strategy is based on

the identification of structural and behavioral deadlock patterns within the dependency cycles identified

in the model. Deadlock risk is broken down into two factors: structural or intrinsic deadlock

risk and behavioral or dynamic deadlock risk.

The process we implemented in DREAM to assess deadlock risk can be summarized as follows:

1. Find all dependency cycles in architectural diagrams where two or more tasks are involved;

2. Search all the structural patterns present in the risky cycles previously identified;

3. Mark all the building elements involved in risky cycles with structural patterns as risky

elements;

4. Assign a numeric value to the intrinsic deadlock risk: the amount of risky cycles containing

structural patterns;

5. Search all the behavioral patterns present in the CFAs where the risky elements participate;

and

6. Assign a numeric value to the behavioral deadlock risk: the amount of risky cycles containing

behavioral patterns.

The first step of this process was implemented through the particularization of a cycle detection

algorithm applicable to undirected graphs with a depth first search strategy. More details about this

algorithm were presented in a previous paper [23].

The results from the cycle detection tool are:

\_ List of cycle sequences.

\_ List of elements involved in the cycles.

Once the cycles are identified, DREAM takes into account two additional contributions to deadlock

risk: structural patterns and behavioral patterns. The structural patterns described in the previous

section are searched in all the dependency cycles. Once a structural pattern is identified, the cycle,

including all the elements participating in the pattern, is marked as risky.