Deadlock Risk Assessment in Architectural Models of Real-Time Systems

Antonio Monzón (antonio.monzon@casa.eads.net)
José-Luis Fernández-Sánchez (jlfdez@etsii.upm.es)
Deadlock?
Why is deadlock still relevant?

◆ It is a subtle problem difficult to identify in design-time (it occurs in execution-time)
◆ It is potentially catastrophic
◆ It is traditionally handled at final stages of the system production lifecycle (SW implementation in fact)
◆ Correction of deadlock may be complex and can lead to other undesirable situations (e.g. race condition)
◆ Deadlock risk can be detected at early stages of system development
◆ RT system design can be driven to minimise deadlock risk
How can deadlock be handled?

- Ignore it...
  - For many soft (or non) RT applications it is not necessary to deal with deadlock

- Prevent it...
  - Simply breaking any of the necessary and sufficient conditions prevents deadlock to happen, but ensuring this may be in most cases expensive (in terms of resources), inefficient or simply non-realistic (e.g. program constraints)

- Avoid it...
  - Making the system deterministic or using resource access protocols allows avoiding deadlock (e.g. careful resource allocation)

- Detect and recover...
  - Create the system to detect when deadlock takes place in execution time and establish mechanisms to get it back to normal operation condition
Why a new algorithm for deadlock?

- The classical four strategies dealing with deadlock assume that information detail is enough to assess deadlock occurrence.
- New assumption: early design stages
  - There is few information detail available to assure whether deadlock will take place in fact or not.
- What is this new approach for?
  - I am interested to know if my architecture is risky with respect to deadlock or not.
  - I am interested to quantify deadlock risk of an architecture with respect to another (in order to choose the less risky).
  - I want to prevent my design to have deadlock risk before I enter to detail it, using some kind of deadlock risk detection.
Does make sense a 'McCabe' for deadlock?

- Cyclomatic complexity is a simple index widely used in industry to measure SW development effort.
- Projects request handy parameters to easily make decisions (excess of formalism is typically rejected by project managers and architects).
- Parameters proposed here are McCabe-fashion.
- Deadlock Risk Index fits at early design stages.
- Complementary techniques to handle deadlock can be used at later stages of the design (or even decide to prevent from deadlock).
Research Work Objectives

- The main objective of this work is to propose a technique to detect the potential occurrence of deadlock with no need to generate software code, estimate execution times or assign priorities.

- In addition some engineering guidelines to minimise the likelihood of deadlock are proposed.

- Finally a tool to automatically assess the deadlock risk of a system design was created.
This paper summarises part of the tasks performed in the development of a doctoral thesis.

Thesis context is the verification of Platform Independent Models (PIM) according to the general Model Driven Architecture (MDA) subject.

The technique proposed is oriented to preliminary analysis of models with very few details about the final solution.

Static Analysis of dependencies and precedence relations of activities using resources are under the scope of this work.
Deadlock Definition

◆ A set of tasks is deadlocked if each task in the set is waiting for an event that only another task in the set can cause.

◆ In computing world, deadlock refers to a specific condition when two tasks are each waiting for the other to release a resource, or more than two tasks are waiting for resources in a circular chain.

◆ Necessary and sufficient conditions for deadlock are: Mutual Exclusion, Hold-and-Wait, Non Pre-emption and Circular Wait.
Deadlock Conditions

- **Mutual exclusion condition**: When tasks claim for exclusive control of resources, e.g. two or more tasks are trying to use a shared resource, and a mutex for this resource is activated, only one task can use the resource and the rest must **wait** until the active task releases the resource.

- **Hold-and-wait condition**: Tasks already holding resources are permitted to request new resources.

- **Non pre-emption condition**: This condition takes place when only the task holding a resource can **release** it.

- **Circular wait condition**: It takes place when tasks are in a **circular chain** and waiting for a resource held by the next task in the chain.
Design Decisions for Deadlock

- **CP::Deadlock**
- **CP::Design Decision**
  - **DL::Mutual Exclusion**
  - **DL::Non-preemption**
  - **DL::Hold & Wait**
  - **DL::Circular Waiting**

- Occurs only if
- Depends on:
  - **CP::Resource Constraint**
  - **CP::Coordination Protocol**
  - **CP::Task Dependency**
Deadlock in PPOOA Structure View

**DEADLOCK RISK**
Deaclock in PPOOA Dynamic View (CFA)
Deadlock Characterisation

- Mutual exclusion condition is represented in PPOOA architectural diagrams by semaphores (mutexes) protecting resources.
- Circular waiting condition is represented in PPOOA architectural diagrams by a dependency cycles.
- Non-preemption condition is implicit in semaphores and buffers coordination protocols.
- Hold-and-wait condition is characterised through some deadlock patterns in CFA diagrams (e.g. task-semaphore-buffer sequence).
Deadlock Assessment Algorithm

1. Search dependency cycles in architectural diagrams with protected resources and buffers.

2. For each risky cycle, obtain the structural complexity. Then characterize the model structural complexity as the maximum value.

3. Analyze the CFAs information to find sequence patterns about the type of dependency of each task considered risky from deadlock perspective.

4. When each task in a cycle holds at least one resource and wants another, or some buffers are used, then deadlock may occur.

5. Assign a numeric value to the risk of deadlock based on the amount of risky cycles detected, the structural complexity and the deadlock sequence protocols.
Deadlock Assessment Algorithm Description

-inputs
- Architectural diagrams: building elements and dependencies
- Behavioural diagrams (CFAs): activities and arcs

- Outputs
- Number of cycles containing risky elements
- List of elements involved in the cycles
- Cycle sequences

- Parameters
- CCI = (WT*NT + WS*NS + WB*NB + WR*NR) / NE
- MCI = Max (CCI)
- DSI = (0.8 * NDS + 0.2 * NFE) / NA
- DRI = NC * Max (MCI, DSI)

<table>
<thead>
<tr>
<th>Element Type</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Tasks (NT)</td>
<td>35%</td>
</tr>
<tr>
<td>Number of Semaphores (NS)</td>
<td>35%</td>
</tr>
<tr>
<td>Number of Buffers (NB)</td>
<td>20%</td>
</tr>
<tr>
<td>Number of Resources (NR)</td>
<td>10%</td>
</tr>
</tbody>
</table>

CCI: Cycle Cycle Complexity Index
MCI: Maximum Complexity Index
DSI: Deadlock Sequence Index
DRI: Deadlock Sequence Index
NC: Number of risky Cycles
NE: Number of elements
NA: Number of Arcs (in CFAs)
Case Study Context

- System types in Airbus Military (embedding SW):
  - Mission: non safety critical SW, multi-user concurrent comms
  - Avionics: from soft-RT to hard-RT safety critical SW (RTCA DO178B–DAL C)
  - Fly-by-wire: hard-RT safety critical SW (DO178B–DAL A)

- Selected case study:
  - Autotuning function within A400M on-board communications management computer
  - Safety critical DAL C avionics
  - ARINC 653 partitioning required by certification authority
  - Case study model involved 3 structural and 7 CFA diagrams
Case Study: Autotuning Function (1)
Case Study: Autotuning Function (2)

04. Triggered Event

- **T1**: PP_Manage_Autotuning_Plan
- **S3**: Semaphore 3
- **B1**: B_Active_Autotuning_Plan
- **S2**: Semaphore 2
- **R1**: List_Active_Autotuning_Plan
- **R4**: Conditions_Checker
- **S1**: Semaphore 1
- **R3**: Stack_Triggered_Event

Flowchart details:
- New conditions
- Request new constraints
- Acquire
- Receive
- Release
- Acquire
- Read list item
- Compare
- Agreed?
  - Yes: Set flag
  - No: Push triggered event
- Acquire
- Release
- Goto next item
- Release
- Goto next item
- Release
### Case Study: Results Discussion

<table>
<thead>
<tr>
<th>Architectural Diagram</th>
<th>Initial Model</th>
<th>Model Alternative</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><img src="image" alt="Diagram" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>Initial Model</th>
<th>Model Alternative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Elements</td>
<td>26</td>
<td>25 (-3.8%)</td>
</tr>
<tr>
<td>Number of Arcs</td>
<td>24</td>
<td>20 (-16.7%)</td>
</tr>
<tr>
<td>Number of Cycles</td>
<td>7</td>
<td>1 (-85.7%)</td>
</tr>
<tr>
<td>MCI</td>
<td>3.7%</td>
<td>3.7% (-)</td>
</tr>
<tr>
<td>NDS</td>
<td>4</td>
<td>0 (-100%)</td>
</tr>
<tr>
<td>NFE</td>
<td>1</td>
<td>1 (-)</td>
</tr>
<tr>
<td>DSI</td>
<td>14%</td>
<td>1% (-93%)</td>
</tr>
<tr>
<td>BDI</td>
<td>14%</td>
<td>3.7% (-26%)</td>
</tr>
<tr>
<td>DRI</td>
<td>0.99</td>
<td>0.01 (-99%)</td>
</tr>
<tr>
<td>Trade-off</td>
<td>Higher deadlock risk</td>
<td>Lower deadlock risk</td>
</tr>
</tbody>
</table>

© Antonio Monzón, 2009  Slide 21/23  http://www.linkedin.com/in/antoniomonzon
Characterisation Trade-off

Benefits
- No execution time estimation is required
- Uses simple parameters
- Easy implementation
- Captures sequence patterns of deadlock (dynamic view)
- Considers cyclic complexity

Drawbacks
- Vague characterisation of structural deadlock patterns (just average complexity parameters)
- Numeric results are difficult to interpret
Conclusions & Future Work

- The proposed approach was useful to make design decisions at early stages of architectural design to minimise the impact of deadlock in detailed design and implementation.

- The proposed approach considered the combination of three parameters to quantify the deadlock risk:
  - Cyclic complexity of the architecture
  - Intrinsic complexity per cycle
  - Sequence deadlock risk

- The results obtained in the application of this approach to a real case study was useful to validate the applicability of the algorithm and to modify the model to reduce the deadlock likelihood.

- Architectural deadlock patterns shall be used to improve the accuracy of the algorithm.

- Validation with additional case studies shall be performed.