Applying Deadlock Risk Assessment in Architectural Models of Real-Time Systems

Antonio Monzón[1], José-Luis Fernández-Sánchez[2], Jorge Ruíz-de-Castañeda[2]

1: Airbus Military, John Lennon Av., 28906 Getafe, Spain
2: Industrial Engineering School, Technical University of Madrid (UPM),
José Gutiérrez Abascal, 2, 28006 Madrid, Spain

Abstract: Software Architectural Assessment is a key discipline to identify at early stages of a real-time system (RTS) synthesis the problems that may become critical in its operation. Typical mechanisms supporting concurrency, such as semaphores or monitors, usually lead to concurrency problems in execution time difficult to identify, reproduce and solve. For this reason it is crucial to understand the root causes of these problems and to provide support to identify and mitigate them at early stages of the system lifecycle. This paper aims to present the results of a research work oriented to the creation of a tool to assess deadlock risk in architectural models of a RTS. A concrete architectural style (PPOOA-UML) was used to represent PIM (Platform Independent Models) of a RTS architecture supported by the PPOOA-Visio CASE tool. A case study was used to validate the deadlock assessment tool created. In the context of one of the functions of a military transport aircraft, the auto-tuning function of the communications system was selected for the assessment of the deadlock risk. According to the results obtained some guidelines are outlined to minimize the deadlock risk of the system architecture.

Keywords: Software Architecture, Real-Time, UML, Concurrency, Deadlock.

1. Introduction

Software architecture modeling is a relevant subject for the production of real-time systems. The development of architectural analysis and design languages in the last years has permitted to represent 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 architecting rules for using these building elements 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 to produce feasible models concerning particular characteristics (e.g. safety). PPOOA architectural style [9] has been selected because it combines both UML notation and MDE concerns, allowing software structural analysis. In addition this style is particularly useful to explicitly represent concurrency issues.

Deadlock is far from being a solved problem. It is in fact an open issue for many research and technical works [2][4][16] and of high interest for industry especially in the real-time embedded domain. Over the last three decades different formal methods have been developed to specify and verify system properties. In this context, Model Checking [5] has become a reference discipline for such approach. Its main goal is to build a finite model (Kripke structure) of a system and check that relevant properties are present in it. What is remarkable from this approach is that an exhaustive search of the state space is performed in order to ensure the property fulfilment. One of the properties particularly checked through model checking techniques applied to concurrent systems is deadlock absence [2][4]. Although the main criticism to this approach is the classical state explosion of the models involved, from a practitioner point of view the main drawbacks are the intrinsic complexity of the modelling techniques and their scalability to large RTS in industrial environments. Industrial applications require simple and practical approaches to be easily adopted.

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 set of resource access protocols known as priority inheritance [14] has as major objective the resolution of priority inversion. As a collateral benefit deadlock is also avoided if priority ceiling (PCP) or highest locker (HLP) protocols are present in the RTOS. The main issue with this mechanism is that very few commercial RTOS support these protocols and their ad-hoc implementation is complex and therefore onerous. Furthermore some authors have reported performance overheads derived from the utilization of these protocols [3].

This paper proposes the implementation of graph theory to characterize the deadlock risk of an architectural model of a RTS. The objective is not to avoid or detect deadlock occurrence or to prove that a design is deadlock-free, but to assess the risk of deadlock present in an architectural model. It is assumed that a model has an intrinsic risk of

deadlock which may condition further design decisions. The idea with this approach is to make designers aware about this risk as soon as possible and with the minimum analysis effort required. The kind of model verification proposed in this paper is static analysis of platform independent models (see Fig. 1).

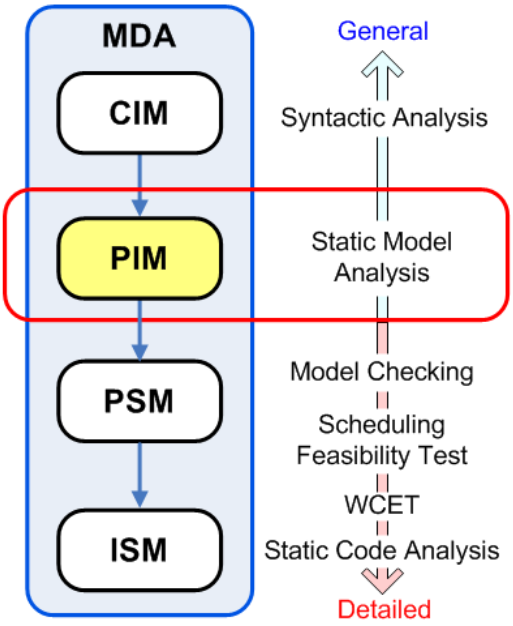


Figure 1: MDA Context

We propose a new control-flow complexity metric based on the properties of the architectural models of a RTS. The information considered in the characterization of deadlock risk comes from the following sources: cyclic complexity of the model and structural and dynamic deadlock patterns. Although the technique proposed is applicable to any kind of system, deadlock topic is of special interest in RT domain.

Although many CASE tools exist in the market to support the software design activities, most of them focus only on the notational aspects with very little concern on engineering activities (e.g. alternative design tradeoffs). Several commercial tools [1][11] support real-time characteristics, but with no specific feature to analyze concurrency problems. Static analysis tools [7] partially cover this topic but their main purpose is to analyse the quality of the software source code (not the models). Finally model-checking tools [2] require the usage of detailed models with formal notations. A clear gap has been identified in current modelling tools to support the concurrency problem assessment at high level of abstraction with semiformal notations (i.e. UML).

In section 2 we briefly describe the PPOOA-UML style fundamentals to better understand the examples of models used later. The proposed characterization of deadlock is presented in section 3. The tool created to automate this assessment is outlined in section 4. And finally sections 5, 6 and 7 discuss about a case study and the alternative design proposed according to the results from the assessment to the concrete RTS model example.

2. PPOOA Architectural Style

A software architectural style encapsulates decisions about its building elements and emphasizes important constraints on the elements and their relations. The PPOOA (Pipelines of Processes in Object Oriented Architectures) architectural style provides building elements for RTS such as components and coordination mechanisms [9]. Constraints on building elements are represented in the PPOOA metamodel and by explicit guidelines. These guidelines not only represent the semantics of the style, they are also helpful for the software architect using the style.

The UML stereotypes are extended with the elements of the PPOOA style (periodic and aperiodic processes, controller objects, and coordination mechanisms). UML Activity diagrams are also adapted for PPOOA style requirements, specifically modelling system responses [8].

The PPOOA architecture diagram is used instead of the UML component diagram to describe the structural view of the RTS architecture. Coordination mechanisms, used as connectors, are also represented in the architecture diagram.

The RTS behaviour view is supported by the "Causal Flow of Activities (CFA)" representation. A CFA represents a behavioural view of the flow of activities performed by the system in response to an event. PPOOA uses the UML activity diagram with partitions to support allocation of activities to the architecture component instances performing them.

For the purposes of this paper, these are the relevant abstractions used in PPOOA for explicit concurrency modelling:

- Task: PPOOA building element representing threads or light processes. It may be periodic or aperiodic.
- Resource: Logical resources can be represented in PPOOA by Domain Components or Structures. These building elements are abstractions of design classes and abstract data types respectively.
- Semaphore: A pure synchronization mechanism. It is the PPOOA building element that supports the synchronization of tasks. Semaphores are used to protect shared logical resources.
- Bounded Buffer: A coordination mechanism representing a FIFO queue used to communicate asynchronously two tasks.

3. Deadlock Characterization

According to Coffman [6], the four necessary and sufficient conditions for deadlock are: Mutual Exclusion, Hold and Wait, Non Pre-emption and Circular Wait.

Circular waiting depends essentially on the tasks interdependency. Tasks must be in a dependency cycle to have a circular waiting. Hold and wait condition depends on the coordination protocol as it describes the way tasks are permitted to access resources. Non-preemption and mutual exclusion conditions may depend on resources constraints and coordination protocols.

In order to highlight the way deadlock is characterized in PPOOA, a theoretical example is used to represent the structural model of a generic system. In Fig. 2, tasks are represented by architectural elements of the type “Process” and resources in the diagram are represented by “Structure” elements of PPOOA style. In this example, the resources are protected by PPOOA semaphores (coordination mechanisms in the style) to guarantee mutual exclusion. This protection involves the first condition for deadlock (mutual exclusion).

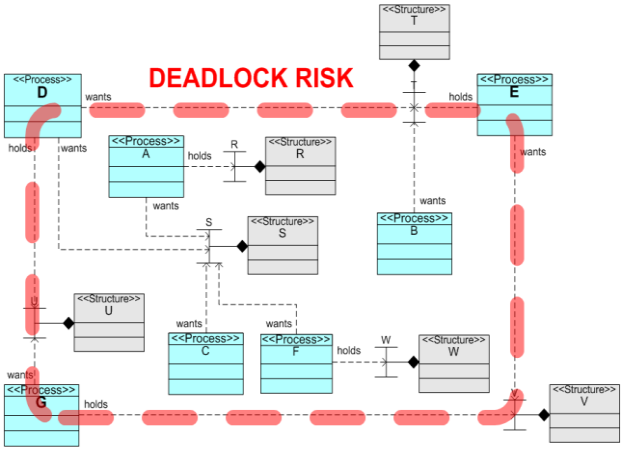


Figure 2. Potential deadlock in the PPOOA architectural view.

Circular waiting condition is represented in the PPOOA architectural view by a dependency cycle. In this case, tasks D, E and G conform a cycle. The rest of tasks in the diagram do not involve any cycle and therefore they cannot contribute to deadlock risk. Non-preemption condition is implicit in the semaphore coordination protocol and cannot be represented diagrammatically.

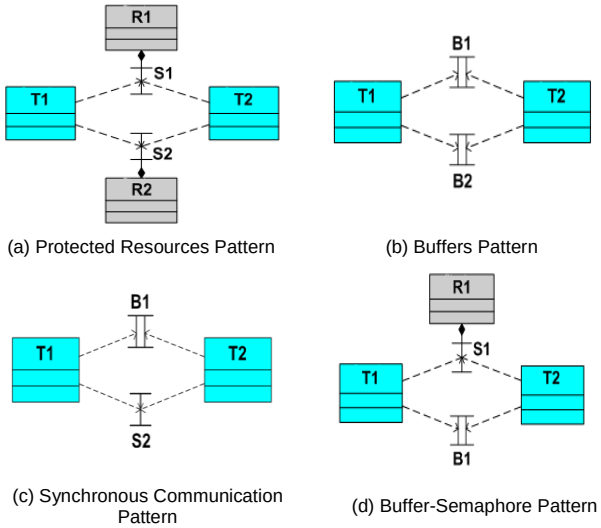
From the circular waiting condition a first criterion for the characterization of deadlock risk can be extracted: identify all dependency cycles where two or more tasks are involved in an architectural model. The reason why two or more tasks are required is that at least two active elements must be competing for shared resources in a dependency cycle. The higher the cyclic complexity of model, the higher its deadlock risk.

Cyclic dependency of several tasks is necessary but not sufficient for deadlock. For the purposes of this paper the rest of conditions shall be summarised as follows: the tasks involved in a dependency cycle must be waiting for conditions depending on other tasks in the same dependency cycle.

The approach proposed in this paper is to refine the cyclic complexity with additional criteria from the structural and behavioural views of an architectural model. This refinement strategy is based on the identification of structural and behavioural deadlock patterns within the dependency cycles identified in the model. Deadlock risk is broken down into two factors: structural or intrinsic deadlock risk, and behavioural or dynamic deadlock risk.

For the structural part of the deadlock risk, four deadlock patterns are proposed, considering the type of constructive elements and the dependency relationships with others in the dependency cycles.

The first structural deadlock pattern (Fig. 3.a) considered in this characterization involves two (or more) tasks and two (or more) resources protected with respective semaphores in the same dependency cycle. This is the classical deadlock case where several tasks are waiting for each other to use locked resources.



(a) Protected Resources Pattern

(b) Buffers Pattern

(c) Synchronous Communication Pattern

(d) Buffer-Semaphore Pattern

Figure 3. Structural Deadlock Patterns.

According to Sutter [16], protected resources are not the only architectural elements susceptible to cause waiting of tasks. Buffers can also introduce some risk of waiting when a task accesses them for some data and they are occasionally empty or full. For this reason buffers can also be considered as risky elements with respect to deadlock. The second structural pattern (Fig. 3.b) involves two (or more tasks) and two (or more) buffers in the same dependency cycle.

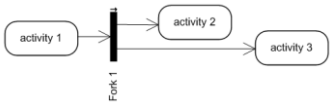
Synchronous message communication can be represented using a combination of a buffer of capacity one and a binary semaphore [10]. This inter-task communication pattern involves task waiting: the producer task waits until the consumer task acknowledges message reception. Therefore it can also be considered risky for deadlock. The third structural pattern (Fig. 3.c) involves two (or more) tasks and one buffer and one semaphore (not protecting resources) in the same dependency cycle.

Finally, for consistency with the two first patterns, a fourth pattern shall be considered (Fig. 3.d): two (or more) tasks, one (or more) buffer and one (or more) semaphore protecting resources in the same dependency cycle.

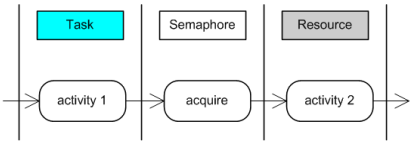
Each time one of these patterns is found in a dependency cycle, the tool records the architectural elements involved and marks them as risky elements from the structural point of view. The dependency cycle where they participate is considered therefore as risky from the structural perspective. The Static Deadlock Risk is defined in this characterization as the total number of risky dependency cycles present in the architectural model.

For the behavioural or dynamic part of deadlock risk, four additional sequence patterns are considered in the behavioural diagrams of the architectural style.

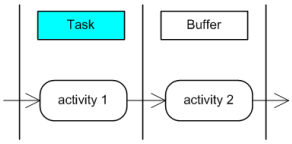
The first pattern (Fig. 4.a) represents the separation of control flow in a CFA. This pattern involves execution parallelism of activities and therefore can be considered risky for deadlock.



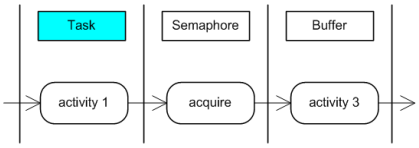
(a) Flow Separation Pattern



(b) Task-Semaphore-Resource Pattern



(c) Task-Buffer Pattern



(d) Task-Buffer-Semaphore Pattern

Figure 3. Behavioural Deadlock Patterns.

The second pattern (Fig. 4.b) is represented by the sequence Task-Semaphore-Resource in a CFA. This pattern is the behavioural counterparty of the first structural pattern. The third behavioural pattern (Fig. 4.c) corresponds to the second structural pattern and the fourth (Fig. 4.d) corresponds to the third structural pattern. The fourth structural pattern has no specific behavioural counterparty because it is in fact considered in the second and third patterns.

Each time one of these sequences is found in a CFA diagram, the tool records the architectural elements involved and checks if they are included in the list of risky elements from the static point of view. In positive case the dependency cycle where they participate is considered therefore as risky from the behavioural perspective.

The overall deadlock risk is characterized by the total number of dependency cycles containing both structural and behavioural deadlock patterns. The interpretation of these parameters is the following: whenever the architectural model of a system has a dependency cycle, there is potential risk of deadlock. This risk is confirmed when those dependency cycles contain structural deadlock patterns. If they do not contain structural deadlock patterns, they can be considered as (conceptually) deadlock-free, with the information available at this stage. Finally the risk is refined with behavioural deadlock patterns. Nevertheless, the absence of behavioural patterns does not guarantee deadlock absence, because this view can be lacking information of the implemented architecture (e.g. sometimes designers consider implicit the participation of semaphores in the access protocol to a protected resource described in a CFA). As a summary, for the purposes of this paper the most relevant evidence of deadlock risk is the existence of dependency cycles containing structural deadlock patterns in structural diagrams. If no risky cycle exists, no deadlock may occur. The characterization proposes a refinement of this parameter based on the sequence information available in the behavioural view.

4. Automatic Deadlock Risk Assessment

The algorithm outline proposed to assess deadlock risk in this paper is 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 behavioural patterns present in the CFAs where the risky elements participate.

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

The first step of this algorithm 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 [12].

The results from the cycle detection tool are:

- List of cycle sequences
- List of elements involved in the cycles

Once the cycles are identified, the tool takes into account two additional contributions to deadlock risk: structural patterns and behavioural patterns. The structural patterns described in previous section are searched in all the dependency cycles. Once a structural pattern is identified the cycle is marked as risky as well as all the elements participating in the pattern. For the behavioural part, all the CFAs are scanned to search each of the patterns identified. This time only those elements considered risky from the structural point of view are considered in the search. Each time a behavioural pattern is found the corresponding CFA is marked as risky. The tool takes into account for the overall behavioural deadlock risk if the elements participating in a risky cycle also participate in a behavioural pattern. In this case the cycle is also marked as risky from the behavioural point of view.

We used PPOOA-Visio tool [13], which is currently supported on the top of Microsoft-Visio®. This tool is flexible enough to extend its functionality to support additional engineering features to assess the concurrency problem identified in this paper. The strategy selected was to use an XML export add-on to generate an intermediate file containing the dependencies and additional information necessary for the tools to assess the models. This tool is conceived to help system architects to assess the deadlock reliability of their designs. But perhaps the most important aspect is that it enables them to compare the relative deadlock risk of several design alternatives, in order to better make and justify formal architectural decisions.

5. Case Study Description

A case study in the field of military avionics is presented here to illustrate the applicability of the proposed deadlock analysis and to validate the assessment tool.

One of the functions provided by the avionics embedded in military aircrafts is the automatic communications management. In particular, the function known as “Automatic Tuning of Communication Equipments” (a.k.a. Autotuning) was selected to illustrate the approach of this paper. This function is in charge of setting the frequencies of all on-board communication equipments (mainly radios and transceivers) in order to avoid unauthorized interception of communications by the enemy.

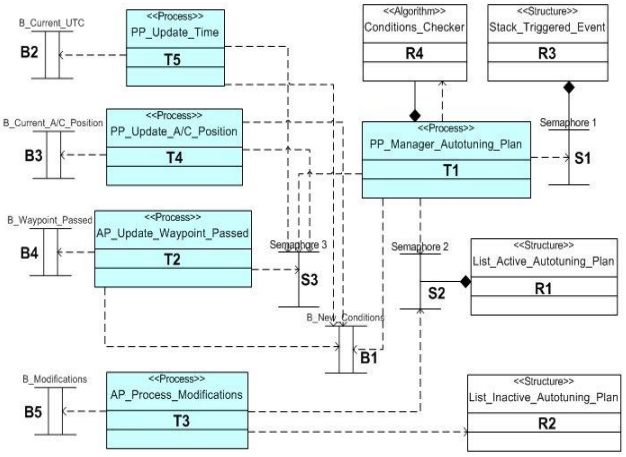


Figure 5. Autotuning Plan Subsystem.

The architectural model for this function was broken down into three subsystems: Autotuning Plan (Fig. 5), Tuning Configuration (Fig. 6) and Autotuning Views Management. This third subsystem was not relevant for this paper.

The Autotuning Plan subsystem captures time, aircraft position and waypoint information from the avionics bus, represented in the architecture by the buffers B2, B3 and B4 respectively. Tasks T2, T4 and T5 are in charge to update these data in the buffer B1. The periodic process T1 (Management Autotuning Plan) is the main task of this subsystem and implements the execution of the autotuning plan. This task browses the list of planned events, implemented by resource R1, and compares them with the queue of events captured in real-time in the buffer B1. It is important to remark that B1 is combined with the binary semaphore S3 (synchronous communication pattern) to ensure that all the messages from the event handlers are received by the plan manager T1.

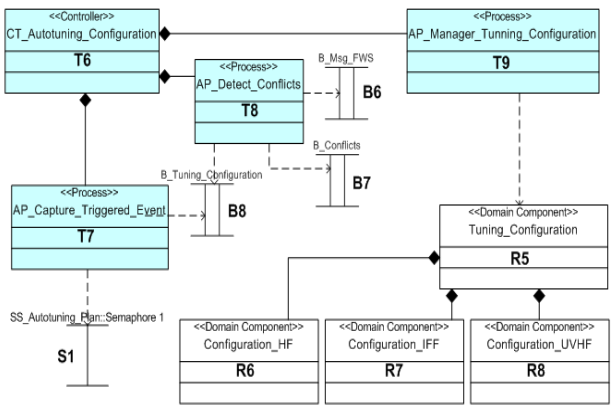


Figure 6. Tuning Configuration Subsystem.

The tasks T1 and T3 can concurrently write on the shared resource R1. In terms of problem domain, the pilot and the planning process can update the autotuning plan. For this reason this resource is protected with another semaphore (S2).

Whenever the planning process T1 detects that a condition takes place, as the result of the periodic comparison, it pushes a new triggering event in the stack R3, which represents the command for the second subsystem to reconfigure the communication equipments tuning.

The aperiodic task T7 captures the event from the protected stack R3 and the controller element Autotuning Configuration sends the command to set new tuning configuration to the communication equipments, represented in the diagram by resources R5, R6, R7 and R8. The process T8 detects conflicts in the configurations of different communication equipments and sends the warning messages through the corresponding bus ports represented by the buffer elements B6 and B7.

This architecture was selected to illustrate the handling of shared resources (it allows parallel execution of tasks), but has some concurrency problems that shall be highlighted in the following section.

The behavioural part of the model is partially represented by the CFA “Triggered Event” (see Fig. 7). This CFA can be interpreted as follows: at the arrival of the internal event “New conditions” the sequence of activities triggered within the system is:

1. T1 (autotuning plan manager) gets new conditions from B1.

2. As long as B1 is protected by S3, before accessing the buffer, the semaphore must be acquired first. In execution time T1 may be forced to remain waiting for the semaphore to be released.

3. Once the buffer is available the message is received by T1.

4. After that the semaphore is released and the list of events opened for reading the next event in it. This list is protected by semaphore S2, and thus T1 can remain waiting until release.

5. T1 compares the information captured from B1 with the one in the list, transformed by the comparison algorithm R4, and if the result is positive the event in the list is flagged and the command for the communication equipments reconfiguration is sent through the intermediate stack R3 (protected by the semaphore S1).

6. Otherwise the next element in the list R1 is analyzed until list end.

6. Results Discussion

The tool transforms the architectural diagrams described in previous section into the aggregated graph shown in Figure 8. This undirected graph represents all the elements of the architecture and the relations among them (regardless their type).

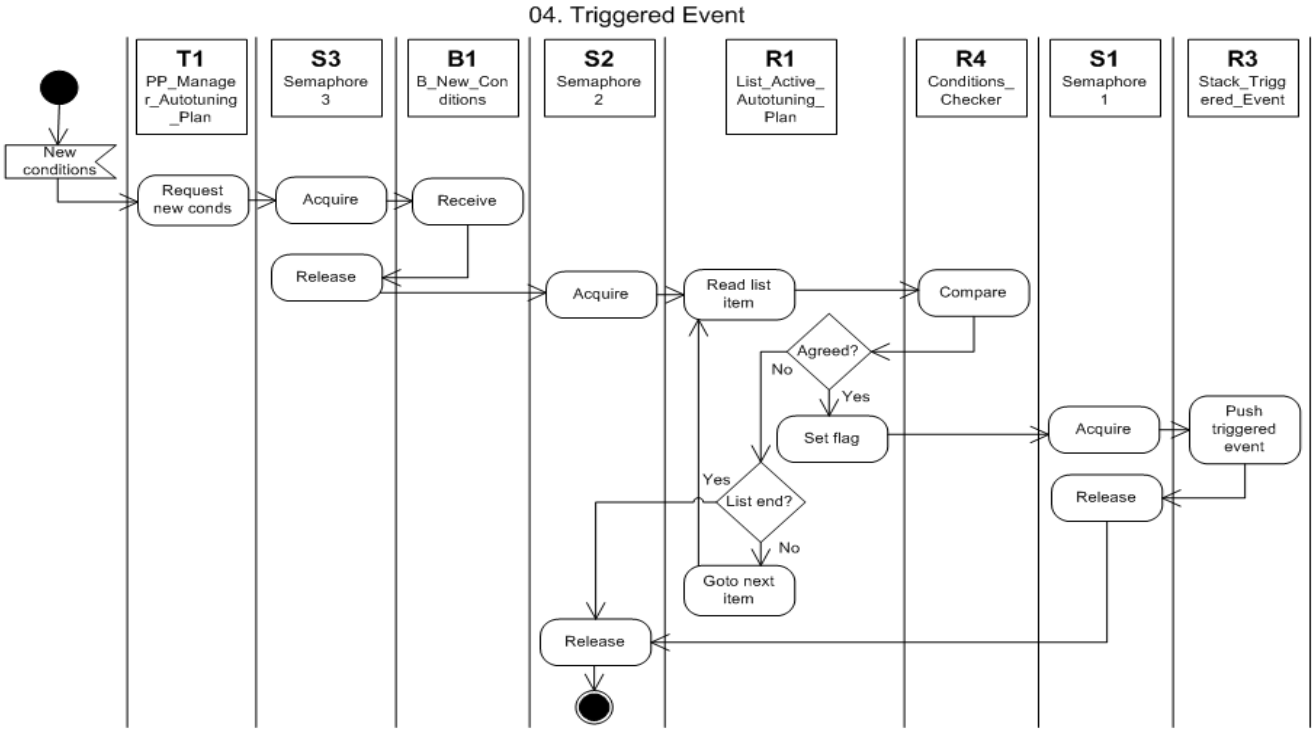


Figure 7. CFA Triggered event.

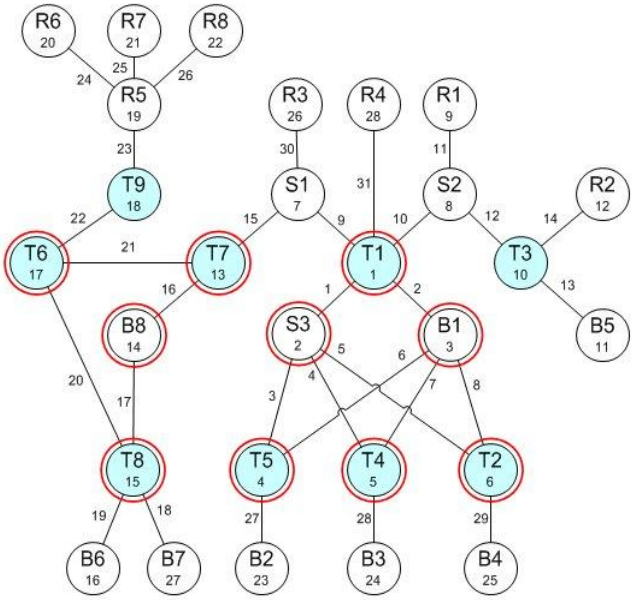


Figure 8. Structural Graph.

This graph is used as input by the cycle identification tool as a first step of the deadlock characterization. The results from this tool are shown in Table 1.

Table 1. List of dependency cycles detected.

Cycle	Elements	Deadlock Risk?
1	PP_Manager_Autotuning_Plan, Semaphore 3, PP_Update_A/C_Position, B_New_Conditions	Yes
2	AP_Update_Waypoint_Passed, Semaphore 3, PP_Update_A/C_Position, B_New_Conditions	Yes
3	PP_Manager_Autotuning_Plan, Semaphore 3, AP_Update_Waypoint_Passed, B_New_Conditions	Yes
4	AP_Update_Waypoint_Passed, Semaphore 3, PP_Update_Time, B_New_Conditions	Yes
5	PP_Manager_Autotuning_Plan, Semaphore 3, PP_Update_Time, B_New_Conditions	Yes
6	PP_Update_A/C_Position, Semaphore 3, PP_Update_Time, B_New_Conditions	Yes
7	Conditions_Checker, PP_Manager_Autotuning_Plan	No
8	AP_Capture_Triggered_Event, B_Tuning_Configuration, AP_Detect_Conflicts, CT_Autotuning_Configuration	No

According to this list, 8 dependency cycles were detected. Out of them only 6 dependency cycles contain two or more tasks with structural deadlock patterns. Therefore the Structural Deadlock Risk (SDR) of this model is 6.

In Figure 8 the elements participating in the structural deadlock patterns are shown in circles. These elements shall be considered risky for deadlock. The risky elements detected are:

- Tasks:
  - PP_Manager_Autotuning_Plan (T1)
  - PP_Update_A/C_Position (T4)
  - AP_Update_Waypoint_Passed (T2)
  - PP_Update_Time (T5)
- Buffers: B_New_Conditions (B1)
- Semaphores: Semaphore 3 (S3)

Once the information from structural diagrams is used, the rest of information relevant for the deadlock analysis comes from the CFAs. In this case study, only three out of seven CFAs included risky elements. The information generated by the tool is shown in Table 2.

Table 2. List of behavioural patterns detected.

#	Task-Semaphore-Buffer	T-B	T-S-R	Separation
1	Analyse waypoint(AP_Update_Waypoint_Passed) -> Acquire(Semaphore 3) -> Send new condition(B_New_Conditions)	None	None	0
2	Analyse time(PP_Update_Time) -> Acquire(Semaphore 3) -> Send time(B_New_Conditions) Analyse position(PP_Update_A/C_Position) -> Acquire(Semaphore 3) -> Send position(B_New_Conditions)	None	None	1
4	Request new conds(PP_Manager_Autotuning_Plan) -> Acquire(Semaphore 3) -> Receive(B_New_Conditions)	None	None	0

From this information the following conclusions can be extracted:

- All the risky elements participate in the risky CFAs 1, 2 and 4.
- All the risky elements participate in at least one behavioural pattern in some risky CFA.
- All the cycles considered risky from the structural point of view are therefore also risky from the behavioural perspective. The Dynamic Deadlock Risk (DDR) of this model is 6.

As a final summary the Table 3 shows the risky elements and the cycles and CFAs where they participate.

7. Alternative Design Discussion and Validation

From the results of previous section the following conclusions can be extracted:

- The elements B_New_Conditions (B1) and Semaphore 3 (S3) are the most conflictive as long as both participate in all risky cycles and CFAs (see Table 3).

Table 3. List of risky elements and their participation in risky cycles and CFAs.

		Risky Cycles						Risky CFAs		
ID	**Risky Element**	**1**	**2**	**3**	**4**	**5**	**6**	**CFA1**	**CFA2**	**CFA4**
1	PP_Manager_Autotuning_Plan (T1)	X		X		X				X
2	PP_Update_A/C_Position (T4)	X	X				X		X	
3	AP_Update_Waypoint_Passed (T2)		X	X	X			X		
4	PP_Update_Time (T5)				X	X	X		X	
5	B_New_Conditions (B1)	X	X	X	X	X	X	X	X	X
6	Semaphore 3 (S3)	X	X	X	X	X	X	X	X	X

- The rest of risky elements participate each in three dependency cycles and one CFA, but all related with the same pattern.
- The elements B1 and S3 are part of the synchronous communication pattern, required to ensure that all the messages sent by the tasks T2, T4 and T5 are received by the plan manager task T1. This pattern is in fact the only source of deadlock risk identified in this case study.

A simple way to reduce the deadlock risk can be to change the current communication pattern among these tasks to an asynchronous pattern. This pattern involves removing the semaphore S3. The resulting model has no risky elements as the semaphore causing the risk is missing. Nevertheless, this design decision is in conflict with the real-time requirement of ensuring that no message from the event managers is lost. For this reason this alternative was discarded.

A second alternative design was proposed in order to fulfil with both requirements: low deadlock risk and reliable message handling. The change consists in splitting the current buffer B_New_Conditions into three buffers each communicating the pairs of tasks: T1-T2, T1-T4 and T1-T5. The results from the deadlock assessment tool show that no risk is present in this alternative model. Further analysis can be performed with temporal data to assess tasks overload with complementary tools like Cheddar.

The validation of the parameters proposed in this paper, as well as the results of the different case studies used to derive them were performed with the aid of the schedulability analysis and simulation tool Cheddar [15], developed in the University of Brest. Cheddar is interoperable with PPOOA-Visio [10]. A specific developed Visio add-on implements the interoperability between PPOOA-Visio and Cheddar, and allows capturing architecture model information generated with PPOOA as an XML file input for Cheddar. Execution times estimation was added to models in order to allow the simulation of execution in Cheddar showing when deadlock occurs.

8. Conclusions and Future Work

This paper proposes a complementary approach to the existing deadlock prevention, avoidance and detection techniques. The automated analysis of basic properties of an architectural model allows the characterization complex problems such as the overall deadlock risk of an RTS architecture with very few design information. The model information used in this characterization is:

- Cyclic dependency of tasks and resources
- Structural patterns in architectural diagrams
- Behavioural patterns in activity diagrams

The appropriate combination of these three sources provides factors (Structural Deadlock Risk and Behavioural Deadlock Risk), which can be used to compute the potential risk of a design to have deadlocks and in addition to compare alternative designs to choose the most appropriate with respect to deadlock.

This analysis has been validated by the application of the tool created to a real case study in the field of military avionics, and the results of the case study were used to make design decisions on alternative designs more reliable concerning deadlock.

Although the tool was created as an add-on on top of PPOOA-Visio, the analysis proposed in this paper can be extended to architectural designs created with any other architectural description language representing explicit concurrency. The conclusions obtained are of general applicability and were considered relevant in practice to make architectural decisions at early stages of the conception of an avionics system.

The following steps of the research work shall be focused in the tuning of the tool with the information extracted from the application to other case study in the field of robotic space missions. The results of these research activities shall be the core of a doctoral thesis. The prototype of the deadlock risk assessment tool has been developed as part of an MsC thesis presented in 2009 in the Technical University of Madrid (UPM).

Finally it should be mentioned that the approach proposed in this paper can be considered as a partial view of a future architecture assessment tool. Additional concurrency and architectural problems in general (e.g. race conditions or low cohesion) could also be addressed with similar approaches. Ideally an engineering dashboard with different indicators of system properties could be created to support architectural decisions and trade-offs.

Acknowledgments

We would like to send our special tanks to Ágatha Puigdueta for her contribution in the description of the problem domain. She works as SW Architect in the SW Avionics Department of Systems Center of Competency of Airbus Military.

For confidentiality reasons no express references have been made to specific project information. All the documentation is under both industrial confidentiality and military security constraints.

All the figures, tables and data shown in this paper were created specifically for the purposes of this paper (none of them appear in any document under Airbus Military copyright).

References

[1] ARTiSAN Software Tools, Inc., Real-Time Studio, http://www.artisansw.com

[2] Bensalem, S., Bozga, M., Nguyen, T., Sifakis, J.: D-Finder: A Tool for Compositional Deadlock Detection and Verification. 21st ICCAV. LNCS, vol. 5643, pp. 614—619. Springer-Verlag, Grenoble, France (July 2009)

[3] Briand, L., Roy, D.: Meeting Deadlines in Hard Real-Time Systems - The Rate Monotonic Approach. IEEE Computer Society Press, 1999.

[4] Chaki, S., Sinha, N.: Assume-Guarantee Reasoning for Deadlock. SEI Technical Note, CMU/SEI-2006-TN-028. (September 2006)

[5] Clarke, E., Grumberg O., Peled, D.: Model Checking. MIT Press, Cambridge, MA, USA. (1999)

[6] Coffman, E. G., Elphick, M. J., Shoshani A.: System deadlocks. Computing Surveys, 3(2):67-78 (June 1971)

[7] Emanuelsson, P., Nilsson, U.: A Comparative Study of Industrial Static Analysis Tools. Technical Reports in Computer and Information Science. Report number 2008:3. Linköping University (January 2008)

[8] Fernandez, J.L., Monzon, A.: Extending UML for Real-Time Component Based Architectures. International Conference on Software & Systems Engineering. Paris (December 2001)

[9] Fernandez, J.L.: An Architectural Style for Object-Oriented Real-Time Systems. Fifth International Conference on Software Reuse. IEEE (1998)

[10] Fernandez, J.L., Marmol, G.: Modelling and Evaluating Real-Time Software Architectures. Ada-Europe 2009. LNCS, vol. 5570, pp. 164--176. Springer-Verlag, Brest, France (2009)

[11] IBM Rational, Rhapsody System Designer, http://www-01.ibm.com/software /awdtools/rhapsody

[12] Monzon, A., Fernandez, J.L.: Deadlock Risk Assessment in Architectural Models of Real-Time Systems. IEEE Symposium on Industrial Embedded Systems (SIES 2009). IEEE Computer Society Press, 2009, pp. 181--190. Lausanne, Switzerland (July 8-10, 2009)

[13] PPOOA-Visio, http://www.ppooa.com.es

[14] Sha, L., Rajkumar, R., J. Lehoczky, Priority Inheritance Protocols: An Approach to Real-Time Synchronization. IEEE Transactions on Computers, Vol. 39, No. 9, 1990

[15] Singhoff, F., Legrand, J., Nana, L., Marcé, L. Cheddar: A flexible real-time scheduling framework. ACM SIGAda Ada Letters, Vol. 24, No. 4, 2004

[16] Sutter, H.: The Many Faces of Deadlock. Dr. Dobbs Journal (August 2008)

Glossary

CASE: Computer Aided Software Engineering

CFA: Causal Flow of Activities

CPU: Central Processing Unit

HLP: Highest Locker Protocol

HMI: Human Machine Interface

MDE: Model Driven Engineering

PCP: Priority Ceiling Protocol

PPOOA: Pipelines of Processes in Object Oriented Architectures

RMA: Rate Monotonic Analysis

RTOS: Real-Time Operating System

RTS: Real-Time System

UML: Unified Modelling Language

XML: Extensible Mark-up Language