Research Journal of Applied Sciences, Engineering and Technology 5(14): 3803-3809, 2013
DOI:10.19026/rjaset.5.4528
ISSN: 2040-7459; e-ISSN: 2040-7467

Submitted: October 12, 2012 Accepted: December 03, 2012 Published: April 20, 2013

Research Article

Ada in Real-time Embedded System

[1]Sima Sinaei, [2]Rabee Sharifi Rad and [3]Elnaz Ghodsvali
[1]Tehran University Tehran, Iran
[2]Sirjan University of Technology, Kerman, Iran
[3]Amir Kabir University Tehran, Iran

Abstract: Ada has an important role in the real-time/embedded/safety-critical areas. It is the only ISO-standard, object-oriented, concurrent, real-time programming language. Ada is used as a usual language for application areas such as defense embedded systems that reliability and efficiency are very essential. One of the main Ada's characteristics in compare with other programming languages is that, Ada was developed from the ground up with capabilities that provide real-time requirements. In this study it will be shown why Ada is used as the new standard for real-time programming languages and basic characteristics of real-time programming system in general and how they are addressed in Ada will be explained.

Keywords: Ada, real-time system, real-time, reliability, scheduing

INTRODUCTION

A Real-time system is a system which process information and the correctness of its functioning depends on the results and the time of producing this result www.adacore.com. In these systems deadline is a significant factor. Practitioners in the field of real-time computer system design often distinguish between hard and soft real-time systems. If missing a deadline leads to complete loss of the task, it is hard real-time and if missing a deadline allows the task to continue by compromising some objectives, it is soft real-time. Facing these tight deadlines can be enhanced by using Ada's real-time standards.

United States Department of Defense realized that billions of dollars was being spent annually on `military' software and especially on the maintenance of real-time and embedded systems. They decided to find a programming language suitable for the department's requirements. The result was Ada. The total number of high-level programming languages in use for such projects fell from over Hundreds in 1983 to Less than forty by 1996. Ada is not limited only for defense contracts anymore and nowadays it is a useful programming language for a lot of different embedded projects such as railway signaling, air traffic control and other usages.

The language became an ANSI standard in 1983 and without any further changes became an ISO standard in 1987. Ada 95, the joint ISO/ANSI standard is another standard for Ada. In 2005, Ada became an ISO standard. The current version of the Ada language standard is known as Ada 2005.

There has been much research in the area of real-time system requirements (Lee and Nehman, 1991). Provides an overview of several issues concerning real-time programming in general and how they are addressed in the primary version of Ada. The primary version was not able to handle some of these issues.

As we mentioned above, Ada improves gradually. By improving Ada and innovation of Ada 2005 as a new standard for real-time applications we consider how these issues would be addressed. This study explains some of basic characteristics of real-time programming and how they implement in the latest version of Ada. It considers if Ada 2005 is enable to solve these shortcomings.

BASIC CHARACTERISTICS OF REAL-TIME PROGRAMMING

Concurrency, Time and Clock Facilities, Scheduling, Facilities for Hardware Control, Priority Inversion and Asynchronous Transfer of Control are some of important issues in the real-time programming (Lee and Nehman, 1991). In each part of this section, these issues will be presented by a short synopsis of its implementation in Ada.

Concurrency: To solve real world problems, it is required programs to exhibit concurrent because the problems in real are inherently concurrent. Therefore Real Time Systems must have high degree of concurrency. The Languages for writing real-time programs should allow multiple threads of control. A

Corresponding Author: Rabee Sharifi Rad, Sirjan University of Technology, Kerman, Iran

process is defined as the notion of single thread of control. Three basic facilities should be provided for process to be able to deal with concurrency: Programming should be done with simultaneous processes, Processes should be synchronized and processes should communicate with each other.

It is possible to classify processes in several topics. We can separate them as independent and cooperating or competing. Independent processes define as a process which do not communicate or synchronize with each other. Cooperating processes do communicate and are synchronous with each other. Competing processes only communicate and synchronize with each other until they are competing for use of a limited number of resources. Processes also classify as static or dynamic. Static process are kind of processes in which the number of processes is fixed but dynamic processes can be created or destroyed at any time.

In Ada, the conventional unit of parallelism, the sequential process, is termed the task. A periodic task is one which has deadlines on a fixed interval; e .g. a task which must read a sensor every Second. Precise scheduling and takes into account how much computation time is needed within each period before the deadline. An aperiodic task is one which is not periodic. Its start time is unpredictable or not regular and it has a certain deadline based on its computational needs. Both kinds of tasks can be built with Ada (Lee and Nehman, 1991).

For expressing concurrent execution four basic mechanisms are possible (Lee and Nehman, 1991):

Co-routines: Co-routines are similar to subroutines, but the control of execution can be transformed explicitly between subroutines non-hierarchically (Bustard, 1990).

Fork and join: This is a simple approach which runs the routines simultaneously with the caller and synchronizes with the created process. This method can be very error prone.

Co-begin and co-end: Co-begin/co-end is a Simple way to describe running of program statements simultaneously.

Explicit declaration: In this approach each process is declared and created explicitly. This allows the structure of the concurrent program to be made clearer.

Ada 2005 proposes the most powerful set of high level concurrency features available in an imperative language and its concurrency semantics is well and precisely defined. Ada allows for the explicit declaration, when the compiled program processes declarations and finds a task declaration it starts execution of the task. Rendezvous is defined as a basic mechanism which let information to exchange. It is used by Ada for both communication and synchronization.

```
Package Ada.Calendar is
-- Standard Ada package, must be
-- supplied with compilers
-- provides useful services for
-- dates and times
Type Time is private;
Subtype Year_Number is Integer range 1901 .. 2099;
Subtype Month_Number is Integer range
1 .. 12;
Subtype Day_Number is Integer range
1 .. 31;
Function Clock return Time;
Function Year (Date: Time) return Year_Number;
Function Month (Date: Time) return Month_Number;
Function Day (Date: Time) return Day_Number;
End Ada.Calendar;
```

Fig. 1: Calendar package

Time and clock facilities: In a real-time system time value and clocks are important in programming. They help the system to interact between concurrent activities and they are so helpful for communication especially with the external environment. Measuring absolute time and relative time are the examples of clock's usage.

During monitoring a process and logging data, it is necessary to know the current date and time and in this situation Absolute time is useful. When passage of time should be measured, for example something should occur 5 time units from now, relative time is useful (Burns, 2009).

A hardware clock can approximate the passage of real-time. In some application programs should be executed with the time of the environment so access to a hardware clock is required.

Ada provides package CALENDAR for accessing a clock function. It is a standard package that allows great flexibility for absolute time. Relative time is handled by subtracting two absolute times. In this package a shortage is faced. An error will occur in the intended time because of the elapsing time during the act of measuring the time and prediction of future time is difficult.

Calendar package which is a language-defined library package is shown in Fig. 1.

A value of Time is consisting of the date and the time of day and time of day is given in second from midnight. Subtype Day-Duration describes second and Day-Duration is defined by means of Duration. The Duration is fixed point type and one of the predefined Scalar types. It is implementation dependent and has a range at least -86,400.0 to +86,400.0 (number of second in a day). Function clock returns the current time. Split and Time-of are subprograms which provide conversion between time and program accessible types. The function Time-Of combines a year number, a month number, a day number and duration, into a value of type time. Conversely, the procedure Split returns all four corresponding values. In addition some arithmetic and

```
Pragma Attach_Handler (Handler,Interrupt);
```

Fig. 2: Pragma for nested style

```
Package Nested_Handler_Example is
 protected type Device_Interface
(Int_ID : Ada.Interrupts.Interrupt_ID) is
 procedure Handler;
 pragma Attach_Handler
 (Handler, Int_ID);
end Device_Interface;
end Nested_Handler_Example;
```

Fig. 3: An example of nested handler

```
Pragma Interrupt  Handler (Handler,Interrupt);
```

Fig. 4: Pragma for non-nested style

Boolean operations are specified. Therefore, an appropriate structure for an abstract data type for time is defined by Package calendar.

Facilities for hardware control: Facilities for hardware control should be provided by the real-time programming languages in order that embedded systems could interact with special-purpose hardware devices for input and output such as sensors, probes, actuators, switches, etc. The two principle mechanisms controlling input/output devices are status polling and interrupt driven. It is preferred for these devices to be interrupt driven rather than polled so the cost of hardware devices can be reduced (Burns and Wellings, 2006).

In Ada an interrupt is considered as hardware generated entry call to a task. The address of the interrupt vector is used as unique identifier for each interrupt.

Ada handles interruption in two styles: Nested and non-nested. In the nested style, existence of the protected object installs an interrupt handler implicitly and when the protected object is not existent the previous treatment is implicitly restored. In the non-nested style, procedure calls install the interrupt handlers explicitly, only when an explicit request is demanded, handlers would be restored.

A handler to be installed in the nested style is identified by the pragma appearing in a protected declaration that shown in Fig. 2.

Handler is the name of a parameter less protected procedure in that protected declaration; Interrupt is an expression of type Interrupt-ID. An example of nested handler is given in Fig. 3 (Wellings and Burns, 2007).

A handler to be installed in the non-nested style is identified by pragma appearing in a protected declaration that shown in Fig. 4.

Again, Handler must be the name of a parameter less protected procedure. As with the Attach-Handler pragma, the protected declaration may not be nested in a subprogram body, task body, or block statement. However, this pragma has an additional restriction: if the protected declaration is for a protected type, objects of that type may not be nested in these places either (Brosgol and Ruiz, 2007).

Scheduling: The most important issue of real-time systems is scheduling of tasks. It is required because each task request must complete it's execution before its deadline.

Process is the fundamental unit of a real-time program for scheduling. A process is defined as the notion of single thread of control. In scheduling each process should terminate on its deadline or before its deadline so it should receive enough computation time.

There are Different kinds of real time schedulers:
On-line/off-line scheduler: if the computation of scheduling and execution time happens simultaneously it is called online scheduler and if scheduling is computed before execution time it is called off-line scheduler.

Static/dynamic priority scheduler: If priority changes at execution time it is called dynamic but static is defined if priority remains stable.

Preemptive or non preemptive scheduler: whether can task can be stopped during its execution time it is called preemptive scheduler otherwise it is non preemptive scheduler (Sha and Goodenough, 1990).

In Ada 2005 real time scheduling mode, it is desired to consider several queues for different priority levels. Each queue contains all tasks with the same priority level and it has a dispatching policy. For scheduling first the highest priority queue with at least one ready task should be selected and then the task to run of the queue should be chosen (Barnes, 2005).

As an Example of the preemptive FIFO-Within-Priorities dispatching Policy When a task becomes ready, it is inserted in the tail of its corresponding priority queue. The task at the head of the queue gets the processor when it becomes the highest ready priority task/queue. When a task becomes blocked or terminated, it leaves the queue and the next task in the queue gets the processor.

The FIFO-Within-Priorities dispatching policy is activated by the code shown in Fig. 5.

But Ada 2005 also provides other dispatching policies: Non -preemptive fixed priority dispatching

```
PragmaTask_Dispatching_Policy
(FIFO_Within_Priorities);
```

Fig. 5: FIFO_within_priorities dispatching

```
Pragma Task_Dispatching_Policy
(Non_Preemptive_FIFO_Within_Priorities);
```

Fig. 6: Non-preemptive fixed priority dispatching

```
Pragma Task_Dispatching_Policy
(EDF_Across_Priorities);
```

Fig. 7: Non earliest deadline first dispatching

```
Pragma Task_Dispatching_Policy
(Round_Robin_Within_Priorities);
```

Fig. 8: Round robin dispatching

```
Pragma Priority_Specific_Dispatching
(FIFO_Within_Priorities, 4, 23);
pragma Priority_Specific_Dispatching
(EDF_Across_Priorities, 2, 3);
pragma Priority_Specific_Dispatching
(Round_Robin_Within_Priorities, 0, 1);
```

Fig. 9: Different dispatching policies

which a task will run until it either blocks itself or completes. It is shown in Fig. 6 (Singhoff, 2007).

Earliest deadline first dispatching in which deadlines and not just priorities are used to dictate which ready task is given the processor. A priority range can be assigned to be governed by the EDF policy. EDF is useful for maximizing system responsiveness, but is less predictable than fixed-priority policies in the presence of overload. It is shown in Fig. 7.

Round robin dispatching is a useful traditional policy when there is a need for fairness in task scheduling. Ready tasks at the highest priority level are time-sliced at a user-specified interval. It is shown in Fig. 8.

Ada 2005 allows a program to use different dispatching policies. Each priority level may have its own dispatching protocol. An example is shown in Fig. 9.

Priority inversion: In a real-time system, it is preferred that the more important tasks execute before the less important ones. Priority inversion is said to occur when a higher-priority task is forced to wait for the execution of a lower-priority task. An Uncontrolled priority inversion is a kind of priority inversion when duration of priority inversion in a system is unlimited. It leads to missing deadlines because of unbounded delays during blocking. It is recognized as a serious problem for real-time systems.

```
Asynchronous_Select:: =
select
Triggering_Alternative
then abort
abortable_part
end select;

Triggering_Alternative:: =
Triggering_statement[sequence_of_st
atements]

Triggering_Statement:: =
Entry_call_statement| delay_satement
abortable_part:: =
Sequence_of_statements
```

Fig. 10: Select_statement

In Ada 95 it is tried to reduce the priority inversion by implementing the Priority Inheritance Protocol (PIP) and in Ada 2005 tried to solve the problem with Priority Ceiling Protocol (PCP). According to PIP, PCP is an extension of PIP, with the added features of preventing deadlocks and priority inversions.

As mentioned above in Ada 95 using The Priority Inheritance Protocol (PIP) which is based on pre-emptive scheduling reduced the priority inversion. The basic idea of PIP is to increase the priority if the lower priority task to that of the highest priority task blocked waiting for that resource when a lower priority task blocks one or higher priority tasks. It's after exiting its critical section, the lower priority task returns to its original priority level. There are two situations in which a high priority task can be blocked by a lower priority task. Firstly, it may be directly blocked in which a higher priority task attempts to lock a locked semaphore. Secondly, a medium priority task can be blocked by a lower priority task. The priority inheritance protocol prevents priority inversion and also bounds blocking times, thus making interacting task sets analyzable (Ruiz, 2007).

PIP alone does not prevent deadlocks. If two tasks attempt to lock two different semaphores in opposite order, a deadlock is formed. This deadlock problem can be solved, if the programmer imposes a total ordering on the semaphore locking. However, there can still be a second problem. The blocking duration for a job, though bounded, can still be substantial, because a chain of blocking can be formed (Jim, 1995).

The Priority Ceiling Protocol (PCP) is similar to PIP and is also based on pre-emptive scheduling. PCP also has the following properties:

- It prevents deadlocks
- It prevents chained blocking, so a high priority task can be blocked by at most one lower priority task, even if the task suspended itself within the critical section.

In addition to the direct blocking and push-through blocking caused by PIP, PCP introduces a third type of blocking, ceiling blocking. Ceiling blocking is needed for the avoidance of deadlocks and chained blocking.

Ada 2005 provides a catalog of features for implementing PCP efficiently. Semaphores can be implemented as protected objects instead of tasks, thus avoiding additional task overhead. Protected objects can be used directly, to provide mutual exclusion (Potratz, 2003).

The common practice in real time applications is to assign priorities to tasks and to use Ceiling-Locking as the locking policy for protected objects in order to avoid the unbounded priority inversion. When using Ceiling-Locking, the priority of the protected object is known as its ceiling priority. For a correct and optimal behavior, the ceiling priority of a protected object should be equal to the highest value among the priorities of all the tasks that use the resource.

React to asynchronous signal: In a real-time system, a task should be able to react immediately to an event. There are specific requirements that need asynchronous signals such as recovery from the fault, changing the mode and using interrupts.

In Ada 2005 an asynchronous SELECT-STATEMENT provides asynchronous transfer of control upon completion of an entry call or the expiration of a delay (Ruiz, 2007). This statement is shown in Fig. 10.

Triggering_statement for the execution of an asynchronous_select can be entry_call_statement or delay_satement. if riggering_statement is an entry_call_statement, the entry_name and actual parameters are evaluated as for a simple entry call and the entry call is issued and if it is delay_satement, the delay_expression is evaluated and the expiration time is determined, as for a normal delay_satement.

if the abortable_part completes before triggering_statement, the program will try to cancel the triggering_statement and if it cancels the asynchronous_select is complete. if the triggering_statement is not canceled, the abortable_part is aborted. if the triggering_statement completes normally, the optional sequence_of_statements of the triggering_alternative is executed after the abortable_part is left.

NOTABLE FEATURE OF ADA

Three facts are very important to choose a language for the real-time system. These facts are Reliability, Safety and Expressiveness. In this part these facts will be consider for Ada 2005.

Reliability: Ada's design was based on Reliability. Specific features include strong typing, checks that prevent buffer overflow, checks that prevent dangling reference, a concurrency feature and an exception handling facility. Ada 2005 enhances this support in several areas: OOP, Read-only parameters, Assertions, Avoidance of race conditions during system initialization, Avoidance of silent task termination (Brosgol and Ruiz, 2007).

Safety: In a programming language it means being able to write programs with high assurance that their execution does not introduce hazards. This translates into language requirements related to program predictability and analyzability in order to allow the system to be certified against safety standards implies several requirements that relate to programming language issues: Predictability and Analyzability. Unfortunately, these requirements conflict with other important goals such as expressiveness and maintainability. Dynamic features and object-oriented programming are examples of conflicts. Ada 2005 addresses these issues in several ways: Language profiles, Ravenscar profile, Safe OOP, Safety-oriented pragmas.

Expressiveness: applications fall across a variety of domains and the programming language or its associated libraries must provide the appropriate functionality. Ada 2005 offers a number of features that increase the language's expressiveness. The following are the new features:

More flexible program structuring: Ada 2005 allows interdependent package specifications, making it easier to model and interface with class libraries as defined in languages such as Java.

Unification of concurrency and OOP: Ada 2005 introduces the concept of a Java-style interface that can be implemented by either a sequential or tasking construct, providing a level of abstraction that is not found in other languages.

New libraries: Ada 2005 adds considerable functionality to the predefined environment. There are new packages, for example, for vectors and matrices, linear algebra and 32-bit character support. A comprehensive containers library provides facilities somewhat analogous to the C++ Standard Template Library.

Improved interfacing: Ada 2005 extends Ada 95's interfacing mechanism, making it easier to construct

programs that combine Ada code with modules from C, C++, or Java.

ADA VERSUS OTHER PROGRAMMING LANGUAGES IN REAL-TIME APPLICATION

In this section Ada is compared with C, C++ and java for embedded systems. Recent enhancements in the new Ada 2005 standard have improved Ada and let Ada to be more manageable for embedded-system developer's job and it can be a better development choice than C, C++ or Java Table 1.

- Shows comparison between them www.adacore.com.

Some of the most important properties which considered in designing Ada are reliability and maintainability and it is tried to improve reliability and maintainability with features that emphasize readability over writability and that detect errors early. Ada's emphasis on readability and reliability does not match with the C family of languages, including C++. Java detects buffer overrun errors, but its weakly typed primitive type facility allows data misuse errors that would be caught in Ada. And unlike both Java and the C-based languages, Ada allows programmers to specify a constrained range for a scalar variable which aids both readability and reliability.

The progression of programming languages has been joined by two major development approaches: procedural programming and object-oriented programming. Some embedded systems can be modeled through a procedural-programming approach; others may best be captured through object orientation in order to facilitate enhancements and maintenance. Ada, like C++, can be used for both procedural and object-oriented programming. C, by contrast, lacks object orientation and purely procedural programming in Java is rather clumsy.

It is also important to remind that Concurrent programming is intrinsically more difficult than sequential programming. Ada has a high-level concurrency model. Many languages (such as C and C++) do not support concurrency directly and instead require the programmer to obtain the desired facilities through libraries. This interferes with portability. Others, most notably Java, have a low-level concurrency mechanism that is error-prone.

Embedded systems often have to perform low-level processing: dealing with storage addresses, laying out data structures with specific fields occurring at specific offsets, querying or specifying the size of data objects, handling interrupts, using specialized hardware instructions, treating data as "untyped" storage elements. All of those capabilities are found in Ada. Moreover and in contrast to C and C++, the Ada rules make it clear to the reader of the program that such system-specific and perhaps potentially unsafe features are being used. Low-level programming in Java requires native code and a corresponding loss of protection.

Table 1: Comparison between Ada, C, C++ and java

	Reliability			
	Ada	C	C++	Java
Strong typing	Yes	Partial	Partial	Partial
Range constraint	Yes	No	No	No
Index checks	Yes	No	No	Yes
Methodologies supported				
Procedural	Yes	Yes	Yes	Awkward
Object orientation	Yes	No	Yes	Yes
Concurrency feature				
Functionality	High	None	None	High
Software engineering	High	Not applicable	Not applicable	Low
Low-level support				
Functionality	High	High	High	Low
Software engineering	High	Low	Low	Low
Subset ability support	High	Low	Low	Low

SUMMARY AND CONCLUSION

Basic characteristic of Real time system are concurrency, Time and Clock Facilities, Facilities for Hardware Control, Scheduling, Priority Inversion and React to asynchronous signal and using Ada would affects them. In some cases the effects are useful and that's why using Ada language is appropriate. Ada was really capable for concurrent programs. It use calendar package for time and clock requirements. For controlling hardware, Ada handles interruption in two styles: nested and non-nested. It considers several queues for different priority levels and can use different scheduling policies for each of them. In ada the priority inversion problem was mostly solved. In Ada 95 it is tried to reduce the priority inversion by implementing the priority inheritance protocol (PIP) and in Ada 2005 tried to solve the problem with Priority Ceiling Protocol (PCP). Ada can react to asynchronous signal well by using Asynchronous select-statement. Notable feature of Ada and comparison of Ada with C, C++ and java was done in this study.

REFERENCES

Barnes, J., 2005. Rationale for Ada 2005: 4 tasking and real-time. Ada User J., 26(3): 1-17.

Brosgol, B. and J. Ruiz, 2007. Ada Enhances Embedded-systems Development. Retrieved from: Embedded.com, WWW website http:// www. embedded. com/ columns/ technical insights/ 196800175?_ requested =167577 (Accessed on: November, 2010).

Burns, A., 2009. Real-time Systems and Programming Languages: Ada, Real-Time Java and C/Real-time POSIX. 4 Edn., Addison-Wesley, New York.

Burns, A. and A. Wellings, 2006. Concurrent and Real-time Programming in Ada 2005. Cambridge University Press, Cambridge.

Bustard, W., 1990. Concepts of Concurrent Programming. SEI-CM-24, Carnegie Mellon University, Software Engineering Institute, April.

Jim, A.R., 1995. Priority Inheritance Protocol in Ada 95. (Research Document), University of Houston, Clear Lake.

Lee, P.N. and W. Nehman, 1991. An overview of real-time issues and Ada. ACM Ada Lett., 11(9): 83-95.

Potratz, E., 2003. A practical comparison between java and Ada in implementing a real-time embedded system. Proceedings of the Annual ACM SIGAda International Conference on Ada (SigAda '03). ACM Press, pp: 71-83.

Ruiz, F., 2007. Ada 2005 for Mission-critical Systems. Retrieved from: www. adacore. com/ uploads/ technical.../Ada05_mission_critical.pdf.

Sha, L. and J.B. Goodenough, 1990. Real-time scheduling theory and Ada. Computer, 23(4): 53-62.

Singhoff, F., 2007. MP1: Real time scheduling theory and its use with ada. Proceedings of the ACM International Conference on SIGAda Annual International Conference (SIGAda '07), pp: 8-8.

Wellings, A.J. and A. Burns, 2007. A framework for real-time utilities for Ada 2005. Proceedings of the 13th International Workshop on Real-time Ada (IRTAW '07), pp: 41-47.