

Information technology — EXTensions for real-time Ada

ICS 35.060

National foreword

This British Standard reproduces verbatim ISO/IEC TR 11735 : 1996 and implements it as the UK national standard.

The UK participation in its preparation was entrusted to Technical Committee IST/5, Programming languages, their environments and systems software interfaces, which has the responsibility to:

- aid enquirers to understand the text;
- present to the responsible international/European committee any enquiries on the interpretation, or proposals for change, and keep the UK interests informed;
- monitor related international and European developments and promulgate them in the UK.

A list of organizations represented on this committee can be obtained on request.

Cross-references

The British Standards which implement international or European publications referred to in this document may be found in the BSI Standards Catalogue under the section entitled 'International Standards Correspondence Index', or using the 'Find' facility of the BSI Standards Electronic Catalogue.

Compliance with a British Standard does not of itself confer immunity from legal obligations.

Summary of pages

This document comprises a front cover, an inside front cover, the ISO/IEC TR title page, pages ii to xiii, a blank page, page 1 to 337 and a back cover.

This British Standard, having been prepared under the direction of the DISC Board, was published under the authority of the Standards Board and comes into effect on 15 April 1997

© BSI 1997

ISBN 0 580 27248 6

Amendments issued since publication

Amd. No.	Date	Text affected

TECHNICAL
REPORT

ISO/IEC
TR 11735

RAPPORT
TECHNIQUE

First edition
Première édition
1996-11-01

**Information technology — EXTensions for real-time
Ada**

**Technologies de l'information — Extensions temps
réel Ada**



Reference number
Numéro de référence
ISO/IEC 11735:1996(E/F)

AVA **Documents**
Document Sharing Hub

Contents

Foreword

Introduction

Section 1: General	1
1.1 Scope	1
1.2 Normative references	1
1.3 Conformance	3
1.3.1 Conformance of an implementation to ISO/IEC 8652	3
1.3.2 Conformance of an implementation to this document	3
1.3.3 Conformance of an application to this document	3
1.3.4 Required documentation	3
1.3.5 Metrics	3
Section 2: Definitions	5
2.1 Conventions	5
2.2 Symbols and abbreviations	5
2.3 Terminology	7
2.4 General terms	7
2.5 General Concepts	9
2.5.1 External objects, logical objects, accessors	9
2.5.1.1 Object names	9
2.5.1.2 Interactions with objects	9
2.5.1.3 Accessors	9
2.5.2 Configuration	11
2.6 Errors	11
Section 3: Distributed and Local Asynchronous Cooperation	13
3.1 Introduction	13
3.1.1 Purpose	13
3.1.2 Asynchronous Cooperation Mechanisms	13
3.1.3 Common features of the packages	15
3.1.3.1 Ada language conformance	15
3.1.3.2 Local and distributed cooperation	15
3.1.3.3 Memory model	17
3.1.3.4 Exceptions	17
3.1.3.5 Initialization	19
3.2 Resources	21
3.2.1 Introduction	21
3.2.2 Resource management	21
3.2.3 Specification of the package EXTRA_Resources	23
3.2.4 Semantic description	25
3.3 Events	29
3.3.1 Introduction	29
3.3.2 Event management	29
3.3.3 Specification of the package EXTRA_Events	31
3.3.4 Semantic description	33

Sommaire

Avant-propos

Introduction

Section 1: Généralités	2
1.1 Domaine d'application	2
1.2 Références normatives	2
1.3 Conformité	4
1.3.1 Conformité d'une implémentation à l'ISO/CEI 8652	4
1.3.2 Conformité d'une implémentation au présent document	4
1.3.3 Conformité d'une application au présent document	4
1.3.4 Documentation requise	4
1.3.5 Métriques	4
Section 2: Définitions	6
2.1 Conventions	6
2.2 Symboles et abréviations	6
2.3 Terminologie	8
2.4 Termes généraux	8
2.5 Concepts généraux	10
2.5.1 Objets externes, objets logiques et désignants	10
2.5.1.1 Noms d'objets	10
2.5.1.2 Interactions avec les objets	10
2.5.1.3 Désignants	10
2.5.2 Configuration	12
2.6 Erreurs	12
Section 3: Coopération asynchrone répartie et locale	14
3.1 Introduction	14
3.1.1 But	14
3.1.2 Les mécanismes de coopération asynchrone	14
3.1.3 Caractéristiques communes aux paquetages	16
3.1.3.1 Conformité au langage Ada	16
3.1.3.2 Coopération locale et répartie	16
3.1.3.3 Modèle mémoire	18
3.1.3.4 Exceptions	18
3.1.3.5 Initialisations	20
3.2 Ressources	22
3.2.1 Introduction	22
3.2.2 Gestion de ressource	22
3.2.3 Spécification du paquetage EXTRA_Resources	24
3.2.4 Description sémantique	26
3.3 Événements à niveau	30
3.3.1 Introduction	30
3.3.2 Gestion d'événements	30
3.3.3 Spécification du paquetage EXTRA_Events	32
3.3.4 Description sémantique	34

Descriptors: data processing, computer software, artificial languages, programming languages, Ada, interfaces, application program interfaces. **Descripteurs:** traitement de l'information, programme de calculateur, langage artificiel, langage de programmation, Ada, interface, interface de programme d'application.

Sommaire

3.4	Pulses	37	3.4	Événements impulsionnels	38
3.4.1	Introduction	37	3.4.1	Introduction	38
3.4.2	Pulse management	37	3.4.2	Gestion d'événements impulsionnels	38
3.4.3	Specification of the package EXTRA_Pulses	39	3.4.3	Spécification du paquetage EXTRA_Pulses	40
3.4.4	Semantic description	41	3.4.4	Description sémantique	42
3.5	Barriers	45	3.5	Barrières	46
3.5.1	Introduction	45	3.5.1	Introduction	46
3.5.2	Barrier management	45	3.5.2	Gestion de barrières	46
3.5.3	Specification of the package EXTRA_Barriers	47	3.5.3	Spécification du paquetage EXTRA_Barriers	48
3.5.4	Semantic description	49	3.5.4	Description sémantique	50
3.6	Buffers	53	3.6	Boîtes à lettres	54
3.6.1	Introduction	53	3.6.1	Introduction	54
3.6.2	Buffer management	53	3.6.2	Gestion de boîtes à lettres	54
3.6.3	Specification of the package EXTRA_Buffers	55	3.6.3	Spécification du paquetage EXTRA_Buffers	56
3.6.4	Semantic description	59	3.6.4	Description sémantique	60
3.7	Blackboards	65	3.7	Tableaux noirs	66
3.7.1	Introduction	65	3.7.1	Introduction	66
3.7.2	Blackboard management	65	3.7.2	Gestion de tableaux noirs	66
3.7.3	Specification of the package EXTRA_Blackboards	67	3.7.3	Spécification du paquetage EXTRA_Blackboards	68
3.7.4	Semantic description	69	3.7.4	Description sémantique	70
3.8	Broadcasts	75	3.8	Diffuseurs	76
3.8.1	Introduction	75	3.8.1	Introduction	76
3.8.2	Broadcast management	75	3.8.2	Gestion de diffuseurs	76
3.8.3	Specification of the package EXTRA_Broadcasts	77	3.8.3	Spécification du paquetage EXTRA_Broadcasts	78
3.8.4	Semantic description	79	3.8.4	Description sémantique	80
3.9	Signals	83	3.9	Signaux	84
3.9.1	Introduction	83	3.9.1	Introduction	84
3.9.2	Specifications of the packages EXTRA_Signal_With_Parameter and EXTRA_Signal	83	3.9.2	Spécifications des paquetages EXTRA_Signal_With_Parameter et EXTRA_Signal	84
3.9.3	Semantic description	83	3.9.3	Description sémantique	84
3.10	Documentation	85	3.10	Documentation	86
Section 4: Specific Input-Output	87		Section 4: Entrées-sorties spécifiques	88	
4.1	Introduction	87	4.1	Introduction	88
4.1.1	Device management	87	4.1.1	Gestion des périphériques	88
4.1.2	Modes of Interaction	89	4.1.2	Modes d'interaction	90
4.1.3	Asynchronous request state automaton	93	4.1.3	Automate d'états des requêtes asynchrones	94
4.1.4	Device state automaton	95	4.1.4	Automate d'états d'un périphérique	96
4.1.5	Specification of the package EXTRA_Devices	97	4.1.5	Spécification du paquetage EXTRA_Devices	98
4.2	Device management services	99	4.2	Services de gestion des périphériques	100
4.2.1	Common services.	99	4.2.1	Services communs	100
4.2.2	Synchronous services	101	4.2.2	Services synchrones	102
4.2.2.1	Specification of the package EXTRA_Devices.Synchronous_Management	101	4.2.2.1	Spécification du paquetage EXTRA_Devices.Synchronous_Management	102
4.2.2.2	Semantic description	103	4.2.2.2	Description sémantique	104
4.2.3	Asynchronous services	107	4.2.3	Services asynchrones	108
4.2.3.1	Specification of the package EXTRA_Devices.Asynchronous_Management	107	4.2.3.1	Spécification du paquetage EXTRA_Devices.Asynchronous_Management	108
4.2.3.2	Semantic description	109	4.2.3.2	Description sémantique	110
4.2.3.2.1	Asynchronous request initiation	109	4.2.3.2.1	Lancement d'une requête asynchrone	110
4.2.3.2.2	Asynchronous request waiting services	111	4.2.3.2.2	Attente d'une requête asynchrone	112
4.2.3.2.3	Asynchronous request cancellation	113	4.2.3.2.3	Abandon d'une requête asynchrone	114
4.3	Data transfer services	115	4.3	Services de transfert de données	116
4.3.1	Synchronous services	115	4.3.1	Services synchrones	116
4.3.1.1	Specification of the package EXTRA_Devices.Synchronous_Transfer	115	4.3.1.1	Spécification du paquetage EXTRA_Devices.Synchronous_Transfer	116
4.3.1.2	Semantic description	117	4.3.1.2	Description sémantique	118
4.3.2	Asynchronous transfer services	119	4.3.2	Services asynchrones	120
4.3.2.1	Specification of the package EXTRA_Devices.Asynchronous_Transfer	119	4.3.2.1	Spécification du paquetage EXTRA_Devices.Asynchronous_Transfer	120
4.3.2.2	Semantic description	119	4.3.2.2	Description sémantique	120

4.4	Specific services	123	4.4	Services spécifiques	124
4.4.1	Specification of the package EXTRA_Devices.Specific	125	4.4.1	Spécification du paquetage EXTRA_Devices.Specific	126
4.4.2	Semantic description	127	4.4.2	Description sémantique	128
4.5	Exceptions in input-output	131	4.5	Exceptions sur les entrées/sorties spécifiques	132
4.5.1	Exception Matrix	133	4.5.1	Matrice d'exceptions	134
4.6	Documentation	135	4.6	Documentation	136
Section 5: Time Management		137	Section 5: Gestion du temps		138
5.1	Introduction	137	5.1	Introduction	138
5.2	Precise durations	139	5.2	Durées précises	140
5.2.1	Purpose	139	5.2.1	But	140
5.2.2	Specifications of the package EXTRA_Precise_Durations	141	5.2.2	Spécifications du paquetage EXTRA_Precise_Durations	142
5.2.3	Semantic description	143	5.2.3	Description sémantique	144
5.2.4	Documentation	143	5.2.4	Documentation	144
5.3	Timers	145	5.3	Sabliers	146
5.3.1	Purpose	145	5.3.1	But	146
5.3.2	Execution of a handler	147	5.3.2	Exécution d'un traitement	148
5.3.3	Specifications of the package EXTRA_Timers	147	5.3.3	Spécifications du paquetage EXTRA_Timers	148
5.3.4	Semantic description	149	5.3.4	Description sémantique	150
5.3.5	Exceptions	153	5.3.5	Exceptions	154
5.3.6	Documentation	153	5.3.6	Documentation	154
5.4	Watchdogs	155	5.4	Chiens de gardes	156
5.4.1	Purpose	155	5.4.1	But	156
5.4.2	Specifications of the generic procedures EXTRA_Watch_Without_Abort and EXTRA_Watch_With_Abort	155	5.4.2	Spécifications des procédures génériques EXTRA_Watch_Without_Abort et EXTRA_Watch_With_Abort	156
5.4.3	Semantic description	157	5.4.3	Description sémantique	158
5.4.4	Documentation	157	5.4.4	Documentation	158
5.5	Logical clocks	159	5.5	Horloges logiques	160
5.5.1	Purpose	159	5.5.1	But	160
5.5.2	Definition of logical clocks	159	5.5.2	Définition de l'horloge logique	160
5.5.2.1	Specification of the package EXTRA_Logical_Clocks	161	5.5.2.1	Spécification du paquetage EXTRA_Logical_Clocks	162
5.5.2.2	Semantic description	165	5.5.2.2	Description sémantique	166
5.5.3	Changing the value of logical clocks	171	5.5.3	Changer la valeur des horloges logiques	172
5.5.3.1	Specification of the package EXTRA_Drifts	173	5.5.3.1	Spécification du paquetage EXTRA_Drifts	174
5.5.3.2	Semantic description	175	5.5.3.2	Description sémantique	176
5.5.4	Changing the time flow of logical clocks	179	5.5.4	Changer l'écoulement du temps des horloges logiques	180
5.5.4.1	Specification of the package EXTRA_Time_Flows	179	5.5.4.1	Spécification du paquetage EXTRA_Time_Flows	180
5.5.4.2	Semantic description	181	5.5.4.2	Description sémantique	182
5.5.5	Documentation	181	5.5.5	Documentation	182
5.6	Ordered time stamps	183	5.6	Estampilles ordonnées	184
5.6.1	Purpose	183	5.6.1	But	184
5.6.2	Specification of the package EXTRA_Ordered_Time_Stamps	183	5.6.2	Spécification du paquetage EXTRA_Ordered_Time_Stamps	184
5.6.3	Semantic description	185	5.6.3	Description sémantique	186
5.6.4	Documentation	185	5.6.4	Documentation	186
5.7	CPU time accounting	187	5.7	Comptabilité du temps CPU	188
5.7.1	Purpose	187	5.7.1	But	188
5.7.2	Specification of the package EXTRA_CPU_Time_Accounting	189	5.7.2	Spécification du paquetage EXTRA_CPU_Time_Accounting	190
5.7.3	Semantic description	191	5.7.3	Description sémantique	192
5.7.4	Documentation	195	5.7.4	Documentation	196

Section 6: Task Dispatching Management	197	Section 6: Gestion de l'ordonnancement des tâches	198
6.1 Introduction	197	6.1 Introduction	198
6.2 Task identities	199	6.2 Identités des tâches	200
6.2.1 Exceptions	199	6.2.1 Exceptions	200
6.2.2 Specification of the package EXTRA_Task_Identification	199	6.2.2 Spécification du paquetage EXTRA_Task_Identification	200
6.2.3 Semantic description	201	6.2.3 Description sémantique	202
6.2.4 Documentation	201	6.2.4 Documentation	202
6.3 Pre-emption control	203	6.3 Contrôle de la préemption	204
6.3.1 Specification of the package EXTRA_Dispatching	203	6.3.1 Spécification du paquetage EXTRA_Dispatching	204
6.3.2 Semantic description	203	6.3.2 Description sémantique	204
6.3.3 Documentation	203	6.3.3 Documentation	204
6.4 Task suspension	205	6.4 Suspension de tâche	206
6.4.1 Specification of the package EXTRA_Asynchronous_Task_Control	205	6.4.1 Spécification du paquetage EXTRA_Asynchronous_Task_Control	206
6.4.2 Semantic description	205	6.4.2 Description sémantique	206
6.4.3 Documentation	205	6.4.3 Documentation	206
6.5 Dynamic priorities	207	6.5 Priorités dynamiques	208
6.5.1 Specification of the package EXTRA_Dynamic_Priorities	207	6.5.1 Spécification du paquetage EXTRA_Dynamic_Priorities	208
6.5.2 Semantic description	207	6.5.2 Description sémantique	208
6.5.3 Documentation	207	6.5.3 Documentation	208
Section 7: Interrupt Management	209	Section 7: Gestion des interruptions	210
7.1 Introduction	209	7.1 Introduction	210
7.1.1 Operations on interrupts	211	7.1.1 Opérations sur les interruptions	212
7.1.2 Execution of an interrupt handler	211	7.1.2 Exécution d'un traite-interruption	212
7.1.3 Exception Invalid_Operation	211	7.1.3 Exception Invalid_Operation	212
7.2 Specification of the package EXTRA_Interrupts	213	7.2 Spécification du paquetage EXTRA_Interrupts	214
7.3 Semantic description	217	7.3 Description sémantique	218
7.4 Documentation	225	7.4 Documentation	226
Section 8: Memory Management	227	Section 8: Gestion de la mémoire	228
8.1 Introduction	227	8.1 Introduction	228
8.2 Memory management	229	8.2 Gestion de la mémoire	230
8.2.1 Operations on different kinds of memories	229	8.2.1 Opérations sur les différentes sortes de mémoires	230
8.2.2 Exceptions	229	8.2.2 Exceptions	230
8.2.3 Configuration	229	8.2.3 Configuration	230
8.3 Specification of the package EXTRA_Memories	231	8.3 Spécification du paquetage EXTRA_Memories	232
8.4 Semantic description	233	8.4 Description sémantique	234
8.5 Documentation	239	8.5 Documentation	240
Annex A : Requirements	241	Annexe A : Expressions de besoin	242
Annex B : Relationship with revised ISO/IEC 8652	275	Annexe B: Relations avec la révision de l'ISO/CEI 8652	276
Annex C : Relationship with other standards	293	Annexe C: Relations avec d'autres normes	294
Annex D : Glossary	300	Annexe D : Glossaire	306
Annex E : Bibliography	314	Annexe E : Bibliographie	314
English alphabetical index	318	Index alphabétique français	328

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The main task of technical committees is to prepare International Standards, but in exceptional circumstances a technical committee may propose the publication of a Technical Report of one of the following types:

- type 1, when the required support cannot be obtained for the publication of an International Standard, despite repeated efforts;
- type 2, when the subject is still under technical development or where for any other reason there is the future but not immediate possibility of an agreement on an International Standard;
- type 3, when a technical committee has collected data of a different kind from that which is normally published as an International Standard ("state of the art", for example).

Technical Reports of types 1 and 2 are subject to review within three years of publication, to decide whether they can be transformed into International Standards. Technical Reports of type 3 do not necessarily have to be reviewed until the data they provide are considered to be no longer valid or useful.

ISO/IEC TR 11735, which is a Technical Report of type 2, was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*.

Annexes A to E of this Technical Report are for information only.

Avant-propos

L'ISO (Organisation internationale de normalisation) et la CEI (Commission électrotechnique internationale) forment le système spécialisé de normalisation mondiale. Les organismes nationaux membres de l'ISO ou de la CEI participent au développement de Normes internationales par l'intermédiaire des comités techniques créés par l'organisation concernée afin de s'occuper des différents domaines particuliers de l'activité technique. Les comités techniques de l'ISO et de la CEI collaborent dans des domaines d'intérêt commun. D'autres organisations internationales, gouvernementales ou non gouvernementales, en liaison avec l'ISO et la CEI participent également aux travaux.

Dans le domaine des technologies de l'information, l'ISO et la CEI ont créé un comité technique mixte, l'ISO/CEI JTC 1.

La tâche principale des comités techniques est d'élaborer les Normes internationales, mais exceptionnellement, un comité technique peut proposer la publication d'un rapport technique de l'un des types suivants:

- type 1, lorsque, en dépit de maints efforts, l'accord requis ne peut être réalisé en faveur de la publication d'une Norme internationale;
- type 2, lorsque le sujet en question est encore en cours de développement technique ou lorsque, pour toute autre raison, la possibilité d'un accord pour la publication d'une Norme internationale peut être envisagée pour l'avenir mais pas dans l'immédiat;
- type 3, lorsqu'un comité technique a réuni des données de nature différente de celles qui sont normalement publiées comme Normes internationales (ceci pouvant comprendre des informations sur l'état de la technique, par exemple).

Les rapports techniques des types 1 et 2 font l'objet d'un nouvel examen trois ans au plus tard après leur publication afin de décider éventuellement de leur transformation en Normes internationales. Les rapports techniques du type 3 ne doivent pas nécessairement être révisés avant que les données fournies ne soient plus jugées valables ou utiles.

L'ISO/CEI TR 11735, rapport technique du type 2, a été élaboré par le comité technique mixte ISO/CEI JTC 1, *Technologies de l'information*.

Les annexes A à E du présent Rapport technique sont données uniquement à titre d'information.

Introduction

The purpose of this Technical Report is to define a standard Ada library for hard real time (HRT) to support application portability at the source level. This is intended for application software developers as well as for Ada real time executive developers.

Organization of this Technical Report

This Technical Report is divided into three elements:

- 1) Statement of scope, normative references and conformance (Section 1)
- 2) Definitions and global concepts (Section 2)
- 3) Various interface facilities (Sections 3 to 8)

Base Documents

The library unit interfaces described in this Technical Report are based on the following documents:

- ExTRA (Extensions Temps-Réel Ada) proposals.
- CIFO 3.0 (Catalogue of Interface Features and Options) developed by the ACM/SIGAda/ARTEWG (Ada RunTime Environment Working Group).

The differences between these documents and this Technical Report are justified by the fact that the editing team has tried to increase as much as possible the global uniformity of this Technical Report and its consistency with other incoming standards, especially Ada9X which aims to provide a revised standard for ISO/IEC 8652:1987.

Background

The developers of this Technical Report represent the major actors of the worldwide hard real time community involved in avionics, aerospace, military, transport and nuclear application areas. In the course of their work, they reviewed related standards and reports. They have focused on the existing practice in the targeted domains to provide uniform solutions for common requirements.

Audience

The intended audience for this Technical Report is all persons concerned with hard real time systems programmed in the Ada language. This includes at least four groups of people:

- (1) Persons buying hard real time systems,
- (2) Persons managing companies involved in the hard real time market,
- (3) Persons implementing real time executives for hard real time systems, and finally
- (4) Persons designing and writing hard real time applications where portability is an objective.

Introduction

Le but du présent Rapport Technique est de définir une bibliothèque Ada normative pour le temps réel à fortes contraintes (HRT) pour supporter la portabilité des applications au niveau du texte source. Elle est destinée aussi bien aux développeurs de logiciels d'application qu'aux développeurs d'exécutifs temps réel Ada.

Organisation du présent Rapport Technique

Le présent Rapport Technique se divise en trois éléments:

- 1) Délimitation du domaine d'application, références normatives and conformité (Section 1)
- 2) Définitions et concepts généraux (Section 2)
- 3) Interfaces des divers services (Sections 3 to 8)

Documents de base

Les interfaces des unités de bibliothèque décrites dans le présent Rapport Technique sont fondées sur les documents suivants:

- Propositions ExTRA (Extensions temps-réel Ada).
- CIFO 3.0 (Catalogue of Interface Features and Options) développé par ACM/SIGAda/ARTEWG (Ada RunTime Environment Working Group).

Les différences entre ces documents et le présent Rapport Technique sont dues au fait que l'équipe d'édition a essayé d'augmenter le plus possible l'uniformité globale du présent Rapport Technique et sa cohérence avec d'autres normes en cours de développement, en particulier avec Ada9X dont le but est de produire une norme révisée à ISO/CEI 8652:1987.

Origine et sources

Les développeurs du présent Rapport Technique représentent les acteurs majeurs de la communauté mondiale du temps réel à fortes contraintes dans les domaines de l'aéronautique, de l'espace, de la défense, du transport et du nucléaire. Au cours de leurs travaux, ils ont examiné les normes et les documents ayant un rapport avec le sujet. Ils se sont concentrés sur la pratique courante dans les domaines visés pour fournir des solutions uniformes à des besoins communs.

Public

Le public visé par le présent Rapport Technique comprend toutes les personnes concernées par les systèmes temps réel à fortes contraintes programmés dans le langage Ada. Cela inclut au minimum les quatre groupes suivants:

- (1) Les personnes qui achètent des systèmes temps réel à fortes contraintes,
- (2) Les personnes qui gèrent des sociétés impliquées dans le marché du temps réel à fortes contraintes,
- (3) Les personnes qui implémentent des exécutifs temps réel pour des systèmes temps réel à fortes contraintes, et enfin
- (4) Les personnes qui conçoivent et écrivent des applications temps réel à fortes contraintes, où la portabilité est un objectif.

Purpose

The principles which have guided the development of this Technical Report are:

The Ada programming language

The Ada programming language was designed in order to build large and complex systems requiring safety, in particular real time embedded systems. This Technical Report is an Ada secondary standard that has been developed to improve economic trade-offs in software for hard real time systems. Some features may be considered for withdrawal in future revisions due to revision of ISO/IEC 8652. These features are retained in this version because of their widespread use or because they are required to define other features.

Within the Ada model

One of the main objectives during the development of this Technical Report was the conformance to the Ada "virtual machine" model keeping the "spirit" of Ada. Special care has been taken to define library units that can be implemented using the Ada programming language.

No interaction with the Ada compiler

One of the main principles that has guided the development of this Technical Report was the necessity to avoid any burden to Ada compiler vendors. Hence, this Technical Report defines only Ada library units and no pragmas or attributes have been added. All the library units can be implemented without any modification to a conformant Ada compiler.

Used and proven, rather than too innovative mechanisms

The developers of this Technical Report have focused on existing practice in the area of hard real time and have tried not to introduce too innovative solutions. Hence, the solutions are to uniformize widely used and proven mechanisms.

Interface, not implementation

This Technical Report defines an interface and not an implementation. No implementation details are given in the normative part of this standard. However, implementation has been constantly present in the minds of the developers in order to make sure that performance requirements can be met and that the library units are implementable across a wide range of existing and future systems and hardware platforms. Some Ada prototypes have been developed for this purpose. A future informative annex will present lessons learned from these prototypes.

Performance

Because hard real time systems are targeted by this Technical Report, performance has always been one of the most important criteria underlying the development of this Technical Report. Simple solutions have been preferred in order to be implemented efficiently. In particular, implementations with static data space configurations can be constructed.

Safety

Safety is one the most important requirement of hard real time systems. Special care has been taken in building this Technical Report to meet this requirement; in particular, whenever possible, mechanisms enabling error recovery and fault tolerance have been included; the semantic models adopted are very simple.

Application oriented

The basic goal was to promote portability of real time applications written in Ada by developing a clear, consistent and unambiguous standard for the interface specifications of a library of real time functionalities. This library can be implemented either in hardware (at least partially) or in portable Ada.

Minimal fully defined interface

The set of library units defined in this Technical Report provides sufficient services, while keeping this standard as small as possible. In particular, redundancy has been avoided. The semantic specifications are precisely and fully defined reducing interpretation ambiguities and implementation dependent behaviors.

Objectif et principes généraux

Les principes qui ont guidé le développement du présent Rapport Technique sont:

Le langage Ada

Le langage de programmation Ada a été conçu pour construire des grands systèmes complexes qui nécessitent un haut degré de sûreté, en particulier dans les systèmes temps réel embarqués. Le présent Rapport Technique est une norme secondaire Ada qui a été développée pour améliorer le bilan économique des logiciels de systèmes temps réel à fortes contraintes. Certaines fonctionnalités de la présente norme pourront être retirées dans les révisions futures, à cause d'une révision de la norme ISO/CEI 8652. Ces fonctionnalités sont gardées dans cette version pour leur large utilisation ou parce qu'elles sont nécessaires pour définir d'autres fonctionnalités.

Dans le modèle Ada

L'un des objectifs principaux pendant le développement de la présente norme a été la conformité au modèle de la "machine virtuelle" Ada, gardant ainsi l'"esprit" Ada. Un soin particulier a été pris pour définir des unités de bibliothèques qui peuvent être implémentées en utilisant le langage de programmation Ada.

Pas d'interaction avec le compilateur Ada

L'un des objectifs principaux qui a guidé le développement de la présente norme a été la nécessité d'éviter de mettre une charge supplémentaire aux fournisseurs de compilateurs Ada. En conséquence, cette norme ne définit que des unités de bibliothèques et ni pragma ni attribut n'ont été ajoutés. Toutes les unités de bibliothèques peuvent être implémentées sans aucune modification d'un compilateur Ada conforme à la norme.

Des mécanismes utilisés et éprouvés, plutôt que des solutions trop innovatives

Les développeurs de cette norme se sont concentrés sur la pratique courante dans le domaine du temps réel à fortes contraintes et ont cherché à ne pas introduire des solutions trop novatrices. En conséquence, les solutions représentent une uniformisation de mécanismes déjà largement utilisés et éprouvés.

L'interface, pas l'implémentation

Cette norme définit des interfaces et non des implémentations. Aucun détail d'implémentation n'est donné dans la partie normative de la présente norme. Cependant, l'implémentation est restée constamment dans les esprits des développeurs afin de s'assurer que les besoins en performance pouvaient être respectés et que les unités de bibliothèque pouvaient être implémentées sur une gamme étendue de systèmes et de plates-formes matérielles présents et futurs. Pour cet objectif, quelques prototypes en Ada ont été développés. Une annexe informative future présentera l'expérience acquise lors de ce prototypage.

Les performances

Parce que les systèmes temps réel à fortes contraintes sont la cible de la présente norme, les performances ont toujours été l'un des critères les plus importants qui ont sous-tendus la construction de cette norme. On a souvent préféré des solutions simples afin qu'elles puissent être implémentées efficacement. En particulier, des implémentations statiques en mémoire peuvent être construites.

La sûreté de fonctionnement

La sûreté de fonctionnement est l'un des besoins les plus importants dans les systèmes temps réel à fortes contraintes. Un soin particulier a été pris dans la construction de la présente norme pour satisfaire ce besoin: en particulier, lorsque cela est possible, il existe des moyens permettant la récupération sur erreur et la tolérance aux fautes; les modèles sémantiques sont des plus simples.

Une orientation vers les applications

A la base, le but a été de promouvoir la portabilité des applications temps réel écrites en Ada en développant une norme claire, cohérente et non ambiguë spécifiant les interfaces d'une bibliothèque de fonctionnalités temps réel. Cette bibliothèque peut être implémentée en matériel (au moins en partie) ou en Ada portable.

Une interface minimale, complètement définie

L'ensemble des unités de bibliothèque qui ont été définies dans la présente norme fournit des services suffisants, tout en gardant la norme la plus minimale possible. En particulier, la redondance a été évitée. Les spécifications sémantiques sont définies précisément et complètement, réduisant ainsi les ambiguïtés d'interprétation et les comportements dépendant des implémentations.

This Technical Report is elaborated by the "Real time Rapporteur Group (RRG)" of the working group on the Ada language of ISO (ISO/IEC JTC1/SC22/WG9).

WG9 convener: Robert Mathis

RRG Project Editor: Nasser KETTANI.

New address : RATIONAL, 1, PI Charles de Gaulle
F-78180 Montigny le Bretonneux , France,
Phone: dir +33 (1) 30 12 09 61, std +33 (1) 30 12 09 50
Fax: +33 (1) 30 12 09 66,
E-mail: kettani@rational.com

Dominique L.R. Chandesris has been the initial project editor of the RRG.

RRG editing team: ExTRA team

The active members of the ExTRA team are:

Brigitte Bauer	<i>SEXTANT AVIONIQUE</i>
Patrick Bougel	<i>DGA/STEI</i>
Michel Coatrieux	<i>THOMSON/CSF</i>
Rémi Creusot	<i>MATRA</i>
Pierre-Yves David	<i>DASSAULT AVIATION</i>
Edmond Devrez	<i>DGA/DCN Ingénierie</i>
Patrick Farail	<i>AEROSPATIALE</i>
Christophe Goethals	<i>DASSAULT ELECTRONIQUE</i>
Georges Joly	<i>DGAC/CENA</i>
Marc Kronental	<i>THOMSON/CSF</i>
Bertrand Leray	<i>DGA/STEI/CELAR</i>

The secretariat of the ExTRA team is assumed by Nasser KETTANI and Vincent CELIER.

Mary Forthofer was a major contributor to correct usage of the English language.

This Technical Report has greatly benefited from cooperation between the ExTRA team and the «Ada Run Time Environment Working Group» (ARTEWG) of the «Special Interest Group on Ada» (SIGAda) of the «Association of Computing Machinery» (ACM) among which some members are important contributors:

Dock Allen	<i>Computing Devices Int'l.</i>
Neal Altman	<i>SEI/CMU</i>
Eric Beser	<i>Westinghouse</i>
George Buchanan	<i>IT Research Institute</i>
Alan Burns	<i>University of York</i>
Scott Carter	<i>McDonnell Douglas Electroni Sys.</i>
Janice Chelini	<i>Delco Systems</i>
Jim Chelini	<i>Raytheon</i>
Patrick Farail	<i>Aerospatiale Aircraft Division</i>
Mary Forthofer	<i>IBM/FSC</i>
Tom Griest	<i>LabTek Corporation</i>
Jeff Grover	<i>GA Tech/GTRI/CSITL</i>
Mark Hall	<i>Lockheed Aeronautical Systems</i>
Juerm Juergens	<i>Encore Computer Corp.</i>
Mike Kamrad	<i>Paramax Systems Corp.</i>
Nasser Kettani	<i>CR2A</i>

Other important contributors are:

Carsten Jurgenson	<i>CRI (Denmark)</i>
Dr Rainer Gerlich	<i>Dornier (Germany)</i>
Jean-Yves Comignion	<i>CR2A (France)</i>
Jean-Marie Sers	<i>Sextant Avionique(France)</i>

Le présent Rapport Technique est élaboré par le "Real time Rapporteur Group" (RRG) du groupe de travail sur le langage Ada de l'ISO (ISO/CEI JTC1/SC22/WG9).

Animateur du WG9: Robert Mathis.

Chef du projet RRG: Nasser KETTANI

Nlle adresse : RATIONAL, 1, PI Charles de Gaulle
F-78180 Montigny le Bretonneux , France,
Phone: dir +33 (1) 30 12 09 61, std +33 (1) 30 12 09 50
Fax: +33 (1) 30 12 09 66,
E-mail: kettani@rational.com

Dominique L.R. Chandesris a été le chef de projet initial. du RRG.

L'équipe d'édition du RRG: Groupe ExTRA

Les membres actifs du groupe ExTRA sont:

Pascal Leroy	<i>SAGEM</i>
Philippe Marquis	<i>DASSAULT AVIATION</i>
Jean-Michel Maurel	<i>SNECMA/Elecma</i>
M. Michel Meignien	<i>THOMSON/CSF</i>
Pierre Parayre	<i>DGA/DEI</i>
Jean Perrin	<i>EDF</i>
Jean Pierre Queille	<i>MATRA Espace</i>
Emeric Rollin	<i>DASSAULT ELECTRONIQUE</i>
Jean-Marie Wallut	<i>CNES</i>
Jean-Michel Tabart	<i>SNECMA/Elecma</i>
Olivier Thivolet	<i>MATRA Défense</i>

Le secrétariat du groupe ExTRA est assuré par Nasser KETTANI et Vincent CELIER.,

Mary Forthofer a été d'une très grande contribution pour la correction de la langue anglaise.

Le présent Rapport Technique a grandement bénéficié de la coopération entre le groupe ExTRA et l'«Ada Run Time Environment Working Group» (ARTEWG) du «Special Interest Group on Ada» (SIGAda) de l'«Association of Computing Machinery» (ACM) dont certains membres sont des contributeurs importants:

Sue Legrand	<i>PRC</i>
Larry Lehman	<i>Integrated Systems Inc.</i>
John Litke	<i>Grumman Corp Res. Center</i>
Carl Malec	<i>Boeing Aerospace</i>
Fred A. Maymir-Ducharme	<i>Paramax-STARs</i>
Charles W. Mckay	<i>University of Houston-</i>
Mike Mills	<i>Department of the Air Force</i>
Richard D. Powers	<i>Texas Instruments</i>
Tom Quiggle	<i>TeleSoft</i>
Roger Racine	<i>Draper Labs</i>
Marc Richard-Foy	<i>Alsys SA</i>
Pat Rogers	<i>SBS Engineering Inc.</i>
John Sniedze	<i>Boeing Defense & Space Group</i>
Chak Sriprasas	<i>Intel Corp.</i>
Joyce Tokar	<i>Tartan, Inc.</i>
Andy Wellings	<i>University of York.</i>

D'autres contributeurs importants sont:

Keith Hopper	<i>(New-Zelande)</i>
Brian Dobbing	<i>Alsys SA (UK)</i>
Daniel Gepner	<i>Dassault Electronique (France)</i>
Dominique Ragueneau	<i>SAGEM (France)</i>

This work is sponsored by the French Ministry of Defense / Armament Board / Electronic and Informatic Direction.

The AJPO (Ada Join Program Office) has given great logistics support to this work, providing public FTP facilities and hosting the RRG mailing list.

If you have interest in participating in the RRG working group, please send your name, address, phone number and potentially electronic mail address to the Secretary, ExTRA, CR2A, 19. Av Dubonnet, F92411 Courbevoie Cedex, France, Phone: +33 (1) 47.68.97.97, Fax: +33 (1) 47.68.87.81, E-mail:extra@CR2A.fr or rrg-request@sw-eng.falls-church.va.us, and ask to be added to the mailing list.

Ce travail est financé par le Ministère français de la Défense / Délégation Générale à l'Armement / Direction de l'Electronique et de l'Informatique.

L'AJPO (Ada Join Program Office) a donné un grand support logistique à ce travail, fournissant des facilités de FTP public et accueillant la liste d'adresses.

Si vous êtes intéressés à participer aux travaux du RRG, veuillez envoyer votre nom, adresse, numéro de téléphone et éventuellement adresse électronique au Secrétariat, ExTRA, CR2A, 19. Av Dubonnet, F92411 Courbevoie Cedex, France, Phone: +33 (1) 47.68.97.97, Fax: +33 (1) 47.68.87.81, E-mail:extra@CR2A.fr ou rrg-request@sw-eng.falls-church.va.us, et demandez à être rajouté dans la liste de diffusion.

Information technology - EXTensions for real-time Ada

Section 1: General

1.1 Scope

- 1 This Technical Report defines standard Ada library units to support application portability at the source level. It is intended to be used both by application software developers as well as Ada real time executive developers.
- 2 This Technical Report is aimed to support development of hard real time embedded applications requiring tight constraints in terms of external environment, time, reliability and safety and that are developed using the Ada programming language standard {1}. The main areas of concerns targeted by this Technical Report are avionics, aerospace, military, nuclear and transport.
- 3 This Technical Report is divided into the following components:
 - (1) **Conventions, terminology, general concepts and definitions**
 - (2) **Definitions of library unit interfaces and specifications of semantics**
- 4 The following areas are outside the scope of this Technical Report:
 - (1) **Interface between the Ada compiler and the Ada run-time system,**
 - (2) **Interface between the Ada run-time system and a potential underlying operating system, and**
 - (3) **Interface of internals of the Ada compiler or run-time system in order to build ancillary tools.**
- 5 This Technical Report describes the external characteristics and facilities that are important to applications developers, rather than the internal techniques employed to achieve these capabilities. This Technical Report has been defined at the source-code level, in the Ada programming language.
- 6 It is outside the scope of this Technical Report to support applications which would use anything other than the Ada programming language together with a conformant implementation of this Technical Report.

1.2 Normatives references

- 1 The following standards contain provisions which, through references in this text, constitute provisions of this Technical Report. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this Technical Report are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/IEC 8652: 1995, *Information technology--Programming languages--Ada.*

Technologies de l'information - Extensions temps réel Ada

Section 1: Généralités

1.1 Domaine d'application

Le présent Rapport Technique définit des unités de bibliothèque Ada normatives pour supporter la portabilité des applications au niveau du texte source. Elle est destinée à être utilisée aussi bien par des développeurs de logiciel d'application que par des développeurs d'exécutifs temps réel Ada. 1

Le présent Rapport Technique est destinée à supporter le développement des applications embarquées temps réel nécessitant des contraintes fortes en termes d'environnement externe, de temps, de fiabilité et de sûreté de fonctionnement, et développées en utilisant le langage de programmation normalisé Ada {1}. Les principaux domaines d'intérêt visés par le présent Rapport Technique sont l'aéronautique, l'espace, la défense, le nucléaire et le transport. 2

Le présent Rapport Technique comprend les éléments suivants: 3

- (1) Conventions, terminologie, concepts généraux et définitions
- (2) Définitions d'interfaces d'unités de bibliothèque et spécifications sémantiques

Les domaines suivants ne sont pas couverts par le présent Rapport Technique : 4

- (1) Interface entre le compilateur Ada et l'exécutif temps réel Ada,
- (2) Interface entre l'exécutif temps réel Ada et un éventuel système d'exploitation sous-jacent, et
- (3) Interface des parties internes du compilateur Ada ou de l'exécutif temps réel Ada pour pouvoir construire des outils de génie logiciel.

Le présent Rapport Technique décrit les caractéristiques externes et les fonctionnalités qui sont importantes pour les développeurs d'application, plutôt que les techniques internes employées pour réaliser ces fonctionnalités. Le présent Rapport Technique a été défini au niveau du code source, dans le langage de programmation Ada. 5

Il est en dehors du domaine d'application du présent Rapport Technique de supporter des applications qui utiliseraient autre chose que le langage de programmation Ada avec une implémentation conforme au présent Rapport Technique . 6

1.2 Références normatives

Les normes suivantes contiennent des dispositions qui, par suite de la référence qui en est faite, constituent des dispositions valables pour le présent Rapport Technique. Au moment de la publication, les éditions indiquées étaient en vigueur. Toute norme est sujette à révision et les parties prenantes d'un accord fondées sur le présent Rapport Technique sont invitées à rechercher la possibilité d'appliquer les éditions les plus récentes des normes indiquées ci-après. Les membres de la CEI et de l'ISO possèdent un registre des Normes Internationales en vigueur à un moment donné. 1

ISO/CEI 8652: 1995, *Technologies de l'information--Langages de programmation--Ada*.

1.3 Conformance

1.3.1 Conformance of an implementation to ISO/IEC 8652

1 An implementation claiming conformance to this Technical Report shall be fully conformant to ISO/IEC 8652.

1.3.2 Conformance of an implementation to this Technical Report

1 A conforming implementation shall support the required interfaces exactly as defined within this Technical Report. These interfaces shall support the functional behavior described herein.

2 Conformance to this document is permitted on a section by section basis.

3 An implementation claiming conformance to a normative section shall support all the required interfaces described in that section. If an implementation supports optional capabilities of this Technical Report, they shall be supported as specified.

4 A conforming implementation is allowed to modify the library unit specifications of this Technical Report in the following ways:

- 5 - add context clauses,
- 6 - add private parts to packages, and
- 7 - modify the visible parts in a way that is neutral to clients.

1.3.3 Conformance of an application to this Technical Report

1 An application is said to be conformant to this Technical Report if it uses the Ada programming language and the library units defined in this Technical Report, and if its execution is conformant to the semantics described in this Technical Report.

1.3.4 Required documentation

1 A *conformance document* with the following information shall be provided as part of a conforming implementation. The conformance document of a conforming implementation shall have the same structure as this Technical Report, with the information presented in the appropriately numbered sections, clauses and subclauses. The conformance document shall not contain any information outside the scope of this Technical Report.

2 The conformance document shall contain a statement that indicates the full name, number and date of the document that applies.

3 Each normative section of this Technical Report defines a documentation that each conforming implementation shall provide. The conformance document shall describe on a section by section basis:

- 4 - the values for all the implementation-defined constants defined in this Technical Report.
- 5 - the behavior of the implementation for all implementation-defined features defined in this Technical Report.
- 6 - the documentation metrics as described in subclause 1.3.5.

7 No library unit specifications other than those described in this Technical Report shall be present in the conformance document.

1.3.5 Metrics

1 Each normative section of this Technical Report defines the documentation of those metrics that each conforming implementation shall provide. The conformance document shall describe, section by section, metrics as defined and required by this Technical Report.

2 The conditions under which the required metrics are obtained should follow the conditions described in this Technical Report. An implementation may provide metrics obtained under other measurement conditions provided that these conditions are described in the conformance document.

1.3 Conformité

1.3.1 Conformité d'une implémentation à l'ISO/CEI 8652

Une implémentation prétendant à la conformité du présent Rapport Technique doit être totalement conforme à l'ISO/CEI 8652. 1

1.3.2 Conformité d'une implémentation au présent Rapport Technique

Une implémentation conforme doit supporter les interfaces requises exactement telles qu'elles sont définies dans le présent Rapport Technique . Ces interfaces doivent avoir le comportement fonctionnel décrit ci-dessous. 1

La conformité au présent Rapport Technique est autorisée section par section. 2

Une implémentation qui prétend à la conformité à une section normative doit fournir toutes les interfaces requises décrites dans cette section. Si une implémentation fournit des fonctionnalités optionnelles du présent Rapport Technique , elles doivent être supportées telles que spécifiées. 3

Une implémentation conforme est autorisée à modifier les spécifications des unités de bibliothèques du présent Rapport Technique des façons suivantes : 4

- ajouter des clauses de contexte, 5
- ajouter des parties privées aux paquetages, et 6
- modifier les parties visibles d'une façon neutre pour les utilisateurs. 7

1.3.3 Conformité d'une application au présent Rapport Technique

Une application est dite conforme au présent Rapport Technique si elle utilise le langage de programmation Ada et les unités de bibliothèques définies dans le présent Rapport Technique , et que son exécution est conforme à la sémantique décrite dans le présent Rapport Technique . 1

1.3.4 Documentation requise

Un *document de conformité* contenant les informations qui suivent doit être fourni par toute implémentation conforme. Le document de conformité d'une implémentation conforme doit avoir la même structure que le présent Rapport Technique , avec les informations présentées dans les sections, articles et paragraphes correctement numérotés. Le document de conformité ne doit contenir aucune information qui soit hors du domaine d'application du présent Rapport Technique . 1

Le document de conformité doit contenir une déclaration indiquant le nom complet, le numéro et la date de la norme qui est appliquée. 2

Chaque section normative du présent Rapport Technique définit une documentation que chaque implémentation conforme doit fournir. Le document de conformité doit décrire, section par section: 3

- les valeurs de toutes les constantes, à définir par l'implémentation, définies dans le présent Rapport Technique. 4
- le comportement de l'implémentation pour toutes les fonctionnalités qui sont à définir par l'implémentation, suivant la définition du présent Rapport Technique. 5
- la documentation des métriques décrite dans le paragraphe 1.3.5. 6

Aucune spécification d'unité de bibliothèque autre que celle décrite dans le présent Rapport Technique ne doit être présente dans le document de conformité. 7

1.3.5 Métriques

Chaque section normative du présent Rapport Technique définit la documentation des métriques que chaque implémentation conforme doit fournir. Le document de conformité doit décrire, section par section, les métriques de bases telles que décrites et requises par le présent Rapport Technique. 1

Il convient que les conditions de mesures de ces métriques suivent les conditions décrites dans le présent Rapport Technique. Une implémentation est autorisée à donner des métriques obtenues dans des conditions de mesures différentes à la condition qu'elle décrive ces conditions dans le document de conformité. 2

Section 2: Definitions

2.1 Conventions

- 1 This Technical Report uses the following conventions to facilitate its reading. Any editorial inconsistencies in using such typographical conventions are not intentional and do not have any normative significance.
- 2 The Ada programming language key words appears in a fixed width font in **bold lower case**.
- 3 Concepts that appears for the first time are in *italic*.
- 4 Ada identifiers use a fixed width font and begin with Upper_Case_Letters.
- 5 Paragraphs numbering follows the style adopted in ISO/IEC 8652.
- 6 Notes are provided for information in order to focus the reader's attention on special points or to explain some choices. Notes are not normative. A note starts with «Note:» and finishes at the end of the paragraph.
- 7 Diagrams are used in order to assist in explaining certain concepts. Such graphics are not normative.
- 8 In general, for each normative section, general concepts are introduced first; Ada specifications and semantic descriptions follow; finally, the required implementation documentation is specified.

2.2 Symbols and abbreviations

2.2.1 The Ada language: This notation is used to refer to ISO/IEC 8652.

2.2.2 ACM: This abbreviation is used to designate «Asynchronous Cooperation Mechanisms».

2.2.3 LRM: This abbreviation is used to designate «Ada programming Language Reference Manual», International Standard ISO/IEC 8652.

Section 2: Définitions

2.1 Conventions

Le présent Rapport Technique utilise les conventions suivantes pour faciliter sa lecture . Des incohérences éditoriales dans l'utilisation de ces conventions typographiques ne sont pas intentionnelles et n'ont aucune signification normative.

Les mots clés du langage Ada apparaissent dans une police à chasse fixe en **minuscule-gras**.

Les concepts qui apparaissent pour la première fois sont en *italique*.

Les identificateurs Ada apparaissent dans une police à chasse fixe et commencent par des Lettres_Majuscules.

Les alinéas sont numérotés suivant le style de la norme du présent Rapport Technique.

Des notes sont fournies à titre informatif pour porter l'attention du lecteur sur un point précis ou pour expliquer un choix. Les notes ne sont pas normatives. Une note commence par «Note:» et se termine à la fin de l'alinéa.

Des diagrammes sont utilisés pour appuyer l'explication de certains concepts. Ces diagrammes ne sont pas normatifs.

En règle générale, pour chaque section normative, les concepts généraux sont d'abord présentés; suivent les spécifications Ada et les descriptions sémantiques; enfin la documentation d'implémentation requise est spécifiée.

2.2 Symboles et abréviations

2.2.1 Le langage Ada: Cette notation est utilisée pour faire référence à l'ISO/CEI 8652.

2.2.2 ACM: Cette abréviation est utilisée pour désigner les "Mécanismes de Coopération Asynchrone".

2.2.3 LRM: Cette abréviation est utilisée pour signifier "Manuel de référence du langage de programmation Ada", Norme internationale ISO/CEI 8652.

2.3 Terminology

1 The form and the tense of verbs used follow the guidelines of annex C of ISO/IEC Directives — Part 3:1989, *Drafting and presentation of International Standards*.

2 The following terms are used throughout this Technical Report. They are introduced here to facilitate the reading of this Technical Report. They are presented in alphabetical order.

2.3.1 Implementation-defined: An indication that an implementation shall define and document a type expression, a data value, a behavior or a feature.

2.3.2 Conformance Document: A document that shall be provided by each implementation and that shall conform to requirements defined in subclause 1.3.4 of this Technical Report.

2.3.3 Supported: A condition regarding optional functionality. Certain functionalities of this Technical Report are optional, but their interface is always required. If the functionality is supported, the associated interfaces work as specified by this Technical Report. If the functionality is not supported, the associated interface shall return the indication specified for that situation. Moreover, the conformance document shall indicate whether or not optional functionalities are supported.

2.4 General terms

For the purposes of this Technical Report, the following general terms apply :

2.4.1 Application: An application is a set of Ada programs that cooperate or not and that execute optionally on different processors. The management of relations between programs that form a single application is made by the configuration.

2.4.2 Configuration: Any computing means that allows the management of system tailoring, system specialization, association between external and logical objects and relation between Ada programs of the same application.

2.4.3 Accessor: An Ada object that allows an Ada program to identify a logical object.

2.4.4 Implementation: The set of hardware (processors, communication tools, physical devices, ...) and software (compilers, link editors, Ada runtimes, predefined library units, ...) that provides for the application of this Technical Report.

2.4.5 Object name: A string that characterizes an external object.

2.4.6 Anonymous object: An external object that does not have its own name; that is, its name is the null string.

2.4.7 External object: Hardware and/or software unit that has a behavior and interaction modes, with Ada programs, defined by this Technical Report.

2.4.8 Logical object: Conceptually, a software object that is created and associated with an external object and that has an accessor.

2.4.9 Program: Notion defined in the ISO/IEC 8652 standard.

2.4.10 Blocked: A task is said to be blocked when, because it is involved in some interaction, it cannot execute any code. Upon activation and prior to its termination, a task is either blocked or ready.

2.4.11 Potentially blocking operation: Any of the following: a **select** statement, an **accept** statement, an entry call statement, a **delay** statement, an **abort** statement, the creation or the activation of a task, a call to a subprogram defined in this Technical Report and specified as a potentially blocking operation, a call to a subprogram whose body contains a potentially blocking operation. In this Technical Report, it is specified for each subprogram if it is or not a potentially blocking operation. A task is not necessarily blocked while executing a subprogram of this Technical Report specified as a potentially blocking operation.

2.4.12 Ready: A task is said to be ready when, if given some execution resource, it can execute some code. Upon activation and prior to its termination, a task is either blocked or ready.

2.4.13 Task dispatching: The process by which one ready task is selected for execution on a processor.

2.3 Terminologie

La forme et le temps des verbes utilisés suivent les dispositions de l'annexe C des Directives ISO/CEI, Partie 3:1989, *Rédaction et présentation des Normes internationales*. 1

Les termes suivants sont utilisés tout au long du présent Rapport Technique. Ils sont introduits ici pour aider à la lecture du présent Rapport Technique. Ils sont présentés dans l'ordre alphabétique. 2

2.3.1 Défini par l'implémentation: Une indication que l'implémentation doit définir et documenter l'expression d'un type, la valeur d'une donnée, un comportement ou une fonctionnalité. 3

2.3.2 Document de conformité: Document devant être fourni par toute implémentation et devant répondre aux exigences définies dans le paragraphe 1.3.4 du présent Rapport Technique. 4

2.3.3 Supporté: Une condition concernant une fonctionnalité optionnelle. Certaines fonctionnalités dans le présent Rapport Technique sont optionnelles, mais leur interface est toujours requise. Si la fonctionnalité est supportée, les interfaces associées fonctionnent comme il est indiqué dans le présent Rapport Technique. Si la fonctionnalité n'est pas supportée, l'interface doit retourner l'indication spécifiée pour cette situation. De plus, le document de conformité doit contenir les indications de support ou non des fonctionnalités optionnelles. 5

2.4 Termes généraux

Pour les besoins du présent Rapport Technique, les termes généraux suivants s'appliquent :

2.4.1 Application: Une application est un ensemble de programmes Ada qui coopèrent ou non entre eux et qui s'exécutent sur des calculateurs éventuellement différents. La gestion des relations entre les programmes formant une même application est faite par la configuration. 1

2.4.2 Configuration: Tout moyen informatique, qui permet de gérer le dimensionnement du système, la spécialisation du système, l'association entre les objets externes et les objets logiques et la relation entre les programmes d'une même application. 2

2.4.3 Désignant: Objet Ada permettant à un programme Ada d'identifier un objet logique. 3

2.4.4 Implémentation: Ensemble du matériel (calculateurs, moyens de communication, périphériques physiques, ...) et du logiciel (compilateurs, éditeurs de liens, exécutifs Ada, unités de bibliothèques prédéfinies, ...) permettant la mise en œuvre du présent Rapport Technique. 4

2.4.5 Nom d'objet: Une chaîne de caractères qui caractérise un objet externe. 5

2.4.6 Objet Anonyme: Objet externe qui n'a pas de nom en propre, c'est-à-dire dont le nom est la chaîne vide. 6

2.4.7 Objet Externe: Unité matérielle et/ou logicielle ayant un comportement et des modes d'interaction, avec les programmes Ada, définis par le présent Rapport Technique. 7

2.4.8 Objet Logique: Conceptuellement, c'est un objet logiciel créé et associé à la création avec un objet externe et possédant un désignant. 8

2.4.9 Programme: Notion définie dans l'ISO/CEI 8652. 9

2.4.10 Bloquée: On dit qu'une tâche est bloquée lorsque, parce qu'elle est impliquée dans une interaction, elle ne peut exécuter aucun code. Après son activation et avant sa terminaison, une tâche est soit bloquée soit prête. 10

2.4.11 Opération potentiellement bloquante: L'une des choses suivantes: une instruction **select**, une instruction **accept**, un appel d'entrée, une instruction **delay**, une instruction **abort**, la création ou l'activation d'une tâche, un appel à un sous-programme de la présente norme spécifiée comme étant une opération potentiellement bloquante, un appel à un sous-programme dont le corps contient une opération potentiellement bloquante. Dans la présente norme, il est spécifié pour chaque sous-programme si c'est une opération potentiellement bloquante ou non. Une tâche n'est pas nécessairement bloquée lorsqu'elle exécute un sous-programme de la présente norme spécifié comme étant une opération potentiellement bloquante. 11

2.4.12 Prête: .i.tâche prête; On dit qu'une tâche est prête quand, si elle dispose de ressources d'exécution, elle peut exécuter du code. Après son activation et avant sa terminaison, une tâche est soit bloquée soit prête. 12

2.4.13 Ordonnancement des tâches: Le processus par lequel une tâche prête est choisie pour s'exécuter sur un processeur. 13

2.5 General Concepts

2.5.1 External objects, logical objects, Accessors

2.5.1.1 Object names

- 1 Applications manage external objects (e.g., devices). Each external object has a name predefined by the implementation. Certain objects may be anonymous; that is, they have the null string as their name.
- 2 Note: There is not necessarily a relation between the name of an object which is used externally by applications and the name which is used internally by the implementation. In particular, anonymous objects may have internal names.
- 3 For each type of external object defined in this Technical Report, the rules for forming names and/or the set of valid names shall be documented by the implementation.

2.5.1.2 Interactions with objects

- 1 Interactions with an external object are expressed as operations on the associated logical object through its accessor. The type of an accessor is called an *accessor type*. All accessor types are private.
- 2 **type *Object* is private;**
- 3 Before being able to interact with an external object, a logical object shall first be associated with it. Logical objects accessors values are returned by the functions *Create*.
- 4 Destroying a logical object (procedures *Destroy*) consists of severing such an association.
- 5 An implementation may set limits on the number of logical objects that a program is permitted to manipulate simultaneously. Such limits shall be documented. Exceeding such limits raises an exception *Limit_Exceeded*.

2.5.1.3 Accessors

- 1 For each accessor type there exists a «null» value that does not designate any logical object. This value is the default value of each accessor variable.
- 2 An accessor type is implementation-dependent. It is conceptually similar to an Ada access type (namely, for the operations *:=*, *=*, */=*).
- 3 Note: This does not necessarily imply dynamic allocation and management of the accessed objects.
- 4 The implicit assignment operation (*:=*) copies the accessor value given by the right-hand expression into the left-hand accessor thus denoting the same object as the one given by the expression.
- 5 The implicit comparison operation (*=*) returns the value *True* if and only if both accessor values are equal (i.e., they denote the same object or both are equal to the «null» value).
- 6 The destroy operation does not update copies (if any) of the accessor used by that destroy operation. The effect of using such copies after destroying the accessed object is to raise an exception.

2.5 Concepts généraux

2.5.1 Objets externes, objets logiques et désignants

2.5.1.1 Noms d'objets

Les logiciels applications manipulent des objets externes (par exemple des périphériques). Tout objet externe a un nom prédéfini par l'implémentation. Certains objets peuvent être anonymes; c'est-à-dire qu'ils ont comme nom la chaîne vide.

Note: Il n'y a pas nécessairement de relation entre un nom utilisé de façon externe par une application et le nom utilisé de façon interne par l'implémentation. En particulier, les objets anonymes peuvent avoir des noms internes.

Pour tout type d'objet externe défini dans la présente norme, les règles d'écriture des noms et/ou l'ensemble des noms valides doivent être documentés par une implémentation.

2.5.1.2 Interactions avec les objets

Les interactions avec un objet externe sont formulées comme des opérations sur un objet logique qui lui est associé, par l'intermédiaire de son désignant. Le type d'un désignant est appelé *type désignant*. Tous les types désignants sont privés.

type Object is private;

Avant de pouvoir interagir avec un objet externe, un objet logique doit d'abord lui être associé. Les valeurs des désignants des objets logiques sont fournies en retour par les fonctions `Create`.

La destruction (procédures `Destroy`) d'un objet logique consiste à rompre cette association.

Une implémentation peut fixer des limites aux nombres d'objets logiques qu'un programme peut manipuler simultanément. Ces limites doivent alors être documentées. Le dépassement de ces limites lève une exception `Limit_Exceeded`.

2.5.1.3 Désignants

Pour chaque type désignant, il existe une valeur "nulle" ne permettant de désigner aucun objet logique. Cette valeur est la valeur par défaut de toute variable de type désignant.

Un type désignant dépend de l'implémentation. Il est conceptuellement similaire au type accès Ada (notamment pour les opérations `:=`, `=`, `/=`).

Note: ceci n'implique pas nécessairement l'allocation et la gestion dynamiques des objets accédés.

L'affectation (`:=`) copie la valeur du désignant donnée par le membre droit de l'expression dans le désignant donné dans le membre gauche se référant ainsi au même objet que celui donné dans l'expression.

L'opérateur implicite de comparaison (`=`) renvoie la valeur `True` si et seulement si les valeurs des deux désignants sont égales (c'est-à-dire qu'ils se réfèrent au même objet ou que tous les deux sont égaux à la valeur "nulle").

Toutes les copies existantes éventuelles d'un désignant ne sont pas mises à jour par l'opération de destruction. L'effet d'une utilisation de telles copies après destruction d'un objet qu'il désigne est de lever une exception.

2.5.2 Configuration

1 Configuration is a means of realizing an implementation of this Technical Report.

2 Configuration means are not specified by this Technical Report. An implementation shall provide, when specified in this Technical Report, configuration means and document their usage.

Notes :

3 1) Hard real time applications, in particular when they are embedded, cannot generally be satisfied merely by compilers and link editors; configuration means are additional means, often necessary, provided by an implementation.

4 2) Configuration may be carried out in various ways: completely frozen, adaptable before link time, adaptable after link time, dynamically adaptable during application execution. The objective is both efficiency and application portability, that is reducing the impact of configuration on application source code. In doing so, a configuration should avoid being frozen (to augment portability) and avoid being dynamic (to augment efficiency).

2.6 Errors

1 The error model defined in this Technical Report is based on Ada exception usage. Hence, each subprogram indicates the occurrence of an error to its callers by raising an exception.

2 Each package of this Technical Report defines a set of exceptions that its subprograms may propagate.

Some packages use common exceptions (e.g., packages of section 3); in such cases, they are defined in a common package and renamed in the user package.

4 Generally, the predefined exceptions of the Ada language are not used in this Technical Report. These exceptions shall not be propagated outside the subprograms of this Technical Report, unless it is explicitly specified.

5 When the state of execution of a subprogram is such that circumstances have arisen where more than one exception could be raised, it shall raise the exception eligible for raising which appears earliest lexically in the semantic description of the subprogram.

2.5.2 Configuration

La configuration est un moyen de mettre en œuvre une implémentation du présent Rapport Technique.

Les moyens de configuration ne sont pas spécifiés par la présente norme. Une implémentation doit fournir, lorsque cela est spécifié dans la présente norme, des moyens de configuration et documenter leur utilisation.

Notes:

1) Les applications temps réel à fortes contraintes, et en particulier embarquées, ne peuvent, en général, se satisfaire uniquement des compilateurs et des éditeurs de liens: les moyens de configuration sont des moyens supplémentaires, souvent nécessaires, fournis par une implémentation.

2) Les moyens de configuration peuvent être de natures différentes: entièrement figés, adaptables avant édition de liens, adaptables après édition de liens, adaptables dynamiquement pendant l'exécution de l'application. L'objectif visé ici est à la fois l'efficacité et la portabilité des applications, c'est-à-dire la réduction de l'impact de la configuration sur le code source des applications. Pour ce faire, une configuration devrait éviter d'être figée (pour augmenter la portabilité) et d'être dynamique (pour augmenter l'efficacité).

2.6 Erreurs

Le modèle d'erreur défini dans la présente norme est fondé sur l'utilisation des exceptions Ada. Ainsi tout sous-programme indique à son appelant l'occurrence d'une erreur par la levée d'une exception.

Chaque paquetage de la présente norme définit un ensemble d'exceptions que ses sous-programmes peuvent propager.

Certains paquetages utilisent des exceptions communes (par exemple, les paquetages de la section 3); elles sont dans ce cas définies dans un même paquetage commun et renommées dans le paquetage utilisateur.

En général, les exceptions prédéfinies du langage Ada ne sont pas utilisées par la présente norme. Ces exceptions ne doivent pas être propagées à l'extérieur des sous-programmes de la présente norme, sauf cas explicitement spécifiés.

Lorsque l'exécution d'un sous-programme est telle que les circonstances sont réunies où plus d'une exception pourrait être levée, celui-ci doit lever l'exception susceptible d'être levée citée en premier dans la description sémantique du sous-programme.

Section 3: Distributed and local asynchronous cooperation

3.1 Introduction

3.1.1 Purpose

The specifications defined in this section provide a full set of Asynchronous Cooperation Mechanisms and have the three following objectives:

- To define a uniform set of asynchronous services that fit the identified needs for functionality, performance and reuse in hard real-time systems.
- To allow efficient implementations of these services on top of available Ada technologies by adapting (optimizing «by hand») their bodies to the underlying Ada Run-time Executives or to hardware solutions.
- To allow for graceful transition from distributed cooperative multiprogramming (i.e., several separate Ada programs cooperate using Asynchronous Cooperation Mechanisms) toward transparent distribution.

These objectives match the following rationale:

- Uniformity of available services promotes the portability of Ada real-time applications design and programming,
- These specifications should not force existing Ada run-time executive vendors to completely rework their product to implement efficiently the defined mechanisms.

Thus, these specifications have been designed using Ada abstractions and Ada semantics such that they can be implemented with Ada tasking facilities.

These services represent a consistent model of asynchronous cooperation mechanisms that promote clean, efficient application architectures which avoid usage of non-portable solutions.

They share a common memory model, which is introduced and discussed in 2.5.1.

3.1.2 Asynchronous Cooperation Mechanisms

This section contains a set of package specifications for various asynchronous cooperation mechanisms.

Six of these mechanisms can be classified into three different classes: counters, states and pulses.

Each class contains two different mechanisms:

- one providing synchronization (without communication), and
- one providing synchronization and communication (through an associated typed message).

The packages are:

- for counters: packages EXTRA_Resources and EXTRA_Buffers,
- for states: packages EXTRA_Events and EXTRA_Blackboards,
- for pulses: packages EXTRA_Pulses and EXTRA_Broadcasts.

In addition, the package EXTRA_Barriers provides a pure synchronization mechanism.

Finally, two generic procedures (EXTRA_Signal_With_Parameter and EXTRA_Signal) provide asynchronous entry or procedure calls.

Section 3: Coopération asynchrone répartie et locale

3.1 Introduction

3.1.1 But

Les spécifications définies par la présente section fournissent un ensemble complet de mécanismes de coopération asynchrone et poursuivent les trois objectifs suivants:

- Définir un ensemble uniforme de services asynchrones qui répondent aux besoins exprimés en termes de fonctionnalité, performance et réutilisabilité dans les systèmes temps-réel à fortes contraintes.
- Permettre des implémentations efficaces de ces services sur les technologies Ada existantes en adaptant (optimisation à la main) leurs corps aux exécuteurs Ada sous-jacents ou à des solutions matérielles.
- Faciliter une transition progressive de la multiprogrammation hétérogène répartie (c'est-à-dire où plusieurs programmes Ada distincts coopèrent en utilisant les mécanismes de coopération asynchrone) vers la répartition transparente.

Ces objectifs correspondent aux justifications suivantes:

- L'uniformité des services disponibles sert à promouvoir la portabilité de l'architecture et du code des applications Ada temps-réel,
- Ces spécifications ne doivent pas contraindre les constructeurs d'exécuteurs à refondre complètement leurs produits pour implémenter efficacement les mécanismes définis.

Aussi ces spécifications ont-elles été conçues en utilisant les abstractions et la sémantique du langage Ada de façon à pouvoir être implémentées sur les mécanismes des tâches Ada.

Ces services représentent un modèle cohérent des mécanismes de coopération asynchrone pour promouvoir des architectures d'applications propres et efficaces évitant ainsi l'utilisation de solutions non-portables.

Ils partagent un modèle commun de gestion mémoire qui est présenté et discuté en 3.1.3.3.

3.1.2 Mécanismes de coopération asynchrone

La présente section contient un ensemble de spécifications de paquetage pour différents mécanismes de coopération asynchrone.

Six de ces mécanismes peuvent être répartis en trois classes: les compteurs, les événements à niveau et les événements impulsionsnels.

Chaque classe contient deux mécanismes différents:

- un mécanisme qui fournit la synchronisation (sans communication), et
- un mécanisme qui fournit synchronisation et communication (par utilisation d'un messages typé)

Les différents paquetages sont:

- pour les compteurs: les paquetages EXTRA_Resources et EXTRA_Buffers,
- pour les événements à niveau: les paquetages EXTRA_Events et EXTRA_Blackboards,
- pour les événements impulsionsnels: les paquetages EXTRA_Pulses et EXTRA_Broadcasts.

De plus, le paquetage EXTRA_Barriers fournit un mécanisme de synchronisation pure.

Enfin, deux procédures génériques (EXTRA_Signal_With_Parameter et EXTRA_Signal) permettent de faire des appels asynchrones à des entrées ou à des procédures.

3.1.3 Common features of the packages

- 1 The entire subclause 3.1.3 applies to the seven following asynchronous cooperation mechanisms: `EXTRA_Resources`, `EXTRA_Events`, `EXTRA_Pulses`, `EXTRA_Barriers`, `EXTRA_Buffers`, `EXTRA_Blackboards`, `EXTRA_Broadcasts`.
- 2 Utilization of the asynchronous cooperation mechanisms between several tasks is done via asynchronous cooperation *objects*. There are as many kinds of objects as there are mechanisms.
- 3 Operations on asynchronous cooperation objects are done using Ada variables or constants called *accessors* which do not directly represent the objects, but provide for their designation (cf 2.5.1).

3.1.3.1 Ada language conformance

- 1 The semantics of asynchronous cooperation mechanisms is strictly conforming to the semantics of the Ada programming language.
- 2 Note: This allows the development of the bodies entirely in Ada.

3.1.3.2 Local and distributed cooperation

- 1 In the remainder of this section, the term *processor* refers to any material entity on which an Ada program may be executed. Such a processor may include several physical processors, with or without a common memory.
- 2 Note: This Technical Report is not concerned with the distribution of an Ada program on several physical processors which is the domain of the Ada compiler.
- 3 Asynchronous cooperation may be either *local* or *distributed*.
 - 4 - it is local when it is achieved only between tasks of the same Ada program,
 - 5 - it is distributed when it is achieved between tasks of different Ada programs
- 6 By extension, a cooperation object is said to be *local*, when it allows only local cooperation, and is said to be *distributed* when it allows distributed cooperation. Thus:
 - 7 - an object is said to be local when it is accessible by a single Ada program,
 - 8 - an object is said to be distributed when it is accessible by several Ada programs
- 9 In the case of distributed cooperation (use of a distributed object), the different Ada programs may be executed on the same processor or on different processors. Communication between these different Ada programs is provided through different hardware or software mechanisms which are not specified by this Technical Report.
- 10 An implementation is authorized to offer only local cooperation. An implementation which offers distributed cooperation shall also offer local cooperation.
- 11 The distinction between the two kinds of cooperation is made during *creation* of a logical object. This creation is done using the function `Create`.
- 12 For each asynchronous cooperation mechanism, there exists a function `Create`, whose profile is the following:


```
13  function Create          (Name          : in String := Anonymous;  
                           <Initialization Information>) return Object;
```
- 14 Each implementation shall indicate the set of authorized values for the parameters `Name` of the different functions `Create` and indicate, if necessary, how distinction is made between local and distributed objects. Any possibility of equivalence between several names shall be documented by the implementation.
- 15 A call to function `Create` returns an accessor.
- 16 A call to function `Create` with the value `Anonymous` for the parameter `Name` always returns an accessor for a new local asynchronous cooperation object. That object is said to be *anonymous*.

3.1.3 Caractéristiques communes aux paquetages

L'ensemble du paragraphe 3.1.3 s'applique aux sept mécanismes de coopération asynchrone suivants: EXTRA_Resources, EXTRA_Events, EXTRA_Pulses, EXTRA_Barriers, EXTRA_Buffers, EXTRA_Blackboards, EXTRA_Broadcasts.

L'utilisation des mécanismes de coopération asynchrone entre plusieurs tâches se fait à l'aide d'*objets* de coopération asynchrone. Il y a autant de sortes d'objet que de mécanismes.

Les opérations sur les objets de coopération asynchrone se font à l'aide de variables ou de constantes Ada appelées *désignants* qui ne représentent pas directement les objets, mais permettent d'identifier ceux-ci (voir 2.5.1).

3.1.3.1 Conformité au langage Ada

La sémantique des mécanismes de coopération est strictement conforme à la sémantique du langage Ada.

Note: Ceci permet de développer les corps entièrement en Ada.

3.1.3.2 Coopération locale et répartie

Dans la suite de cette section, on appellera *processeur* toute entité matérielle sur laquelle un programme Ada peut s'exécuter. Un tel processeur peut être composé de plusieurs processeurs physiques, avec ou sans mémoire commune.

Note: Le présent Rapport Technique ne s'intéresse pas à la répartition d'un programme Ada sur plusieurs processeurs physiques, qui reste du domaine du compilateur Ada.

La coopération asynchrone peut être soit *locale* soit *répartie*.

- elle est locale lorsqu'elle n'est faite qu'entre tâches d'un même programme Ada.
- elle est répartie lorsqu'elle est faite entre des tâches de programmes Ada différents.

Par extension, on dira qu'un objet de coopération est *local* lorsqu'il ne permet que la coopération locale, et qu'il est *réparti*, lorsqu'il permet la coopération répartie. De ce fait:

- un objet est dit local lorsqu'il n'est accessible que par un seul programme Ada,
- un objet est dit réparti lorsqu'il est accessible par plusieurs programmes Ada.

Dans le cas d'une coopération répartie (utilisation d'un objet réparti), les différents programmes Ada peuvent s'exécuter sur le même processeur ou sur des processeurs distincts. La communication entre ces différents programmes Ada s'effectue par des mécanismes tant matériels que logiciels qui ne font pas l'objet du présent Rapport Technique.

Une implémentation est autorisée à ne fournir que la coopération locale. Si une implémentation fournit la coopération répartie, elle doit également fournir la coopération locale.

La distinction entre les deux genres de coopération se fait au moment de la *création* d'un objet logique. Cette création se fait en utilisant la fonction Create.

Pour chaque mécanisme de coopération asynchrone, il existe une fonction Create dont le profil est le suivant:

```
function Create (Name : in String := Anonymous;
                Informations d'Initialisation ) return Object;
```

Chaque implémentation doit indiquer l'ensemble des valeurs possibles pour les paramètres Name des différentes fonctions Create et indiquer, si nécessaire, comment se fait la distinction entre les noms des objet locaux et des objet répartis. Toute possibilité d'équivalence entre plusieurs noms doit être documentée par l'implémentation.

Une fonction Create fournit en retour un désignant.

Une fonction Create avec la valeur Anonymous comme paramètre Name fournit toujours en retour un désignant sur un nouvel objet de coopération asynchrone local. Cet objet est dit *anonyme*.

In the other cases, in the same Ada program, two calls to the same function `Create` with the same value for the parameter `Name`, or with two equivalent values, return the same accessor. However, in the same Ada program, two calls to `Create` of two instantiations of the same generic package, with the same value for the parameter `Name` or with two equivalent values, are forbidden.

For each type of asynchronous cooperation object, a constant `Null_Accessor` is defined:

```
Null_Accessor      : constant Object;
```

All asynchronous cooperation object accessors are initialized by default to `Null_Accessor`.

For each asynchronous cooperation mechanism, there exists a procedure `Destroy` whose profile is the following:

```
procedure Destroy   (An_Object : in out Object);
```

The effect of this procedure is to *delete* the asynchronous cooperation object identified by parameter `An_Object` and to set the value of this parameter to `Null_Accessor`. If the parameter `An_Object` does not identify an existing object, this procedure has no other effect than to set the value of `An_Object` to `Null_Accessor`.

For each asynchronous cooperation mechanism, there exists a `Name_Of` function which returns the *name* of the object (value equivalent to the parameter `Name` of the function `Create`) with which the accessor was obtained.

3.1.3.3 Memory model

All asynchronous cooperation mechanisms are defined using abstract data types. The asynchronous cooperation objects that may be created with these types are handled through *accessors* (cf 2.5.1).

In some implementations, the parameter `Name` of the function `Create` may be used to denote statically pre-allocated buffers handled in a specific area configured externally (e.g., with a run-time executive configuration tool). The semantics of the parameter `Name` shall be documented by each implementation.

In the same Ada program, when two accessors refer to the same asynchronous cooperation object, two calls to the function `Name_Of` using these two accessors shall return identical results (for which the equality operator (`=`) will return the value `True`).

Moreover, in an Ada program, a function `Create` for which the value of `Name` is the one returned by a function `Name_Of`, applied to a distributed object already created in another Ada program, shall return an accessor referring to the same object.

The consistency of the asynchronous cooperation objects shall be guaranteed no matter what changes of state are requested.

3.1.3.4 Exceptions

Exceptions may be raised during the utilization of asynchronous cooperation mechanisms.

Some of these exceptions are specific to a particular mechanism, whereas others are common to several or all mechanisms. These last ones are declared in the `EXTRA_ACM_Exceptions` package, whose sole goal is to share these exceptions, like the `IO_Exceptions` package (LRM, chapter 14). ACM is an acronym for Asynchronous Cooperation Mechanism.

The specification of the package `EXTRA_ACM_Exceptions` is the following:

```
package EXTRA_ACM_Exceptions is
```

```
    Inconsistent      : exception;
    Non_Existent      : exception;
    Failure            : exception;
    Invalid_Name       : exception;
    Unreachable        : exception;
    Waiting_Task_Overflow : exception;
    Limit_Exceeded     : exception;
```

```
end EXTRA_ACM_Exceptions;
```

Dans les autres cas, dans un même programme Ada, deux appels à la même fonction `Create` avec la même valeur du paramètre `Name`, ou avec deux valeurs équivalentes, fournissent en retour le même désignant. Cependant, dans un même programme Ada, deux appels à `Create` issus de deux instanciations d'un même paquetage générique, avec la même valeur du paramètre `Name` ou avec deux valeurs équivalentes, ne sont pas autorisés.

Pour chaque type d'objet de coopération asynchrone, une constante `Null_Accessor` est définie:

```
Null_Accessor      : constant Object;
```

Tous les désignants d'objets de coopération asynchrones sont initialisés par défaut à `Null_Accessor`.

Pour tous les mécanismes de coopération asynchrone, il existe une procédure `Destroy` dont le profil est le suivant:

```
procedure Destroy      (An_Object      : in out Object);
```

L'effet de cette procédure est de *détruire* l'objet de coopération asynchrone identifié par le paramètre `An_Object` et de donner à ce paramètre la valeur `Null_Accessor`. Si le paramètre `An_Object` n'identifie pas un objet existant, cette procédure a pour seul effet de donner au paramètre `An_Object` la valeur `Null_Accessor`.

Pour chaque mécanisme de coopération asynchrone, il existe une fonction `Name_Of` qui fournit en retour le *nom* de l'objet (valeur équivalente au paramètre `Name` de la fonction `Create`) avec lequel un désignant a été obtenu.

3.1.3.3 Modèle mémoire

Tous les mécanismes de coopération asynchrone sont définis par des types abstraits de données. Les objets de coopération qui sont créés avec ces types sont manipulés à travers des *désignants* (voir 2.5.1).

Dans certaines implémentations, le paramètre `Name` de la fonction `Create` peut être utilisé pour se référer à des tampons statiquement pré-configurés et manipulés dans un espace spécifique configuré de façon externe (par exemple par un outil de configuration d'exécutif). La sémantique du paramètre `Name` doit être documentée par toute implémentation.

Dans un même programme Ada, lorsque deux désignants se réfèrent au même objet de coopération asynchrone, deux appels à la fonction `Name_Of` sur ces deux désignants doivent fournir en retour des résultats identiques (pour lesquels l'opérateur d'égalité (=) fournit en retour la valeur `True`).

De plus, dans un programme Ada, une fonction `Create` dont la valeur de `Name` est celle fournie par une fonction `Name_Of` appliquée à un objet réparti déjà créé dans un autre programme Ada, doit fournir en retour un désignant se référant au même objet.

La cohérence des objets de coopération asynchrone, quels que soient les changements d'état demandés, doit être garantie.

3.1.3.4 Exceptions

Des exceptions peuvent être levées lors de l'utilisation des mécanismes de coopération asynchrone.

Certaines de ces exceptions sont spécifiques à un mécanisme particulier, tandis que d'autres sont communes à plusieurs ou à tous les mécanismes: ces dernières exceptions ont été rassemblées dans le paquetage `EXTRA_ACM_Exceptions`, dont le but est de les mettre en commun, comme pour `IO_Exceptions` (LRM, chapitre 14). ACM est l'abréviation de l'anglais «Asynchronous Cooperation Mechanism».

La spécification du paquetage `EXTRA_ACM_Exceptions` est la suivante:

```
package EXTRA_ACM_Exceptions is
  Inconsistent      : exception;
  Non_Existent      : exception;
  Failure           : exception;
  Invalid_Name      : exception;
  Unreachable       : exception;
  Waiting_Task_Overflow : exception;
  Limit_Exceeded    : exception;
end EXTRA_ACM_Exceptions;
```

- 5 The exception `Inconsistent` is raised if the initialization parameters of a function `Create` are not compatible with the identified object.
- 6 The exception `Non_Existing` is raised if an attempt is made to use an accessor which does not identify an existing object.
- 7 The exception `Non_Existing` is also propagated to every task waiting on an object when this object is destroyed (through a call to the procedure `Destroy`) by another task.
- 8 Depending on the implementation, the exception `Failure` may be raised when the transmission of a message between a caller and an object (`Buffer`, `Blackboard` or `Broadcast`) did not maintain the integrity of the message (i.e., some bits of the message have been lost).
- 9 The exception `Invalid_Name` is raised if the parameter `Name` of a function `Create` does not identify an asynchronous cooperation object. It is also raised, in the same Ada program, two calls to `Create` of two instantiations of the same generic package (`EXTRA_Buffers`, `EXTRA_Blackboards`, `EXTRA_Broadcasts`), with the same value for the parameter `Name` or with two equivalent values, are performed.
- 10 The exception `Unreachable` is raised if communication between a caller and an object is not possible for the time being.
- 11 The exception `Waiting_Task_Overflow` is raised if an attempt is made to exceed the maximum allowed number of tasks waiting on an object.
- 12 The exception `Limit_Exceeded` is raised if any other limits are explicitly set by the implementation with regard to the number of existing asynchronous cooperation objects and the completion of a `Create` operation would exceed one of these limits.
- 13 All of the asynchronous cooperation mechanism package specifications reexport the exceptions of this package, that may be raised by one of their operations, by renaming them (e.g., `Non_Existing: exception renames EXTRA_ACM_Exceptions.Non_Existing;`).

3.1.3.5 Initialization

- 1 Objects of type `Resource`, `Event`, `Buffer` and `Barrier` may be *initialized* when a call is made to function `Create`. This is indicated by a parameter `Parameters` of type `Initial_Parameters`.
- 2 The types `Initial_Parameters` are records with a discriminant `Change` of type `Boolean`.
- 3 For every type `Initial_Parameters`, there exist two constants: `No_Change` and `Default`.
- 4 The constant `No_Change` corresponds to a value of the discriminant `Change` equal to the value `False`.
- 5 When the discriminant `Change` of the parameter `Parameters` has the value `False` (`No_Change`), an object is not initialized, or it is initialized by default.
- 6 When the discriminant `Change` of the parameter `Parameters` has the value `True`, the following situations may arise:
- 7 - for a statically configured object, the initialization parameters shall be exactly the same as the characteristics of the configuration for the given object, otherwise the exception `Inconsistent` is raised;
 - 8 - for a dynamically created object, when it is not the first call to the function `Create` for the given object the initialization parameters shall be exactly the same as the characteristics of the given object, otherwise the exception `Inconsistent` is raised;
 - 9 - for the first call to function `Create` for a dynamically created object, the newly created object takes the values of the initialization parameters as initial characteristics.
- 10 For a dynamically created object, the initial characteristics of an object newly created by a first call to function `Create` with the value `False` as the discriminant `Change` of the parameter `Parameters` take default values.

L'exception **Inconsistent** est levée lorsque les paramètres d'initialisation d'une fonction **Create** ne sont pas compatibles avec l'objet identifié.

L'exception **Non_Existent** est levée si l'on tente d'utiliser un désignant qui n'identifie pas un objet existant.

L'exception **Non_Existent** est également propagée à toutes les tâches en attente sur un objet quand celui-ci est détruit (par appel de la procédure **Destroy**) par une autre tâche.

Selon l'implémentation, l'exception **Failure** peut être levée lorsque la transmission d'un message entre un appelant et un objet (**Buffer**, **Blackboard** ou **Broadcast**) n'a pas conservé l'intégrité du message (c'est-à-dire que des bits du messages ont été perdus).

L'exception **Invalid_Name** est levée lorsque le paramètre **Name** d'une fonction **Create** ne permet pas d'identifier un objet de coopération asynchrone. Elle est aussi levée si dans un même programme Ada, deux appels à **Create** issus de deux instantiations d'un même paquetage générique (**EXTRA_Buffers**, **EXTRA_Blackboards**, **EXTRA_Broadcasts**), avec la même valeur du paramètre **Name** ou avec deux valeurs équivalentes sont effectués.

L'exception **Unreachable** est levée lorsque la communication entre un appelant et un objet n'est pas possible temporairement.

L'exception **Waiting_Task_Overflow** est levée lorsqu'il y a tentative de dépassement du nombre maximum de tâches en attente sur un objet.

L'exception **Limit_Exceeded** est levée lorsque certaines autres limites ont été explicitement fixées par l'implémentation quant au nombre d'objets de coopération asynchrone existants et que la réussite d'une opération **Create** dépasserait l'une de ces limites.

Toutes les spécifications des mécanismes de coopération asynchrone réexportent les exceptions de ce paquetage, qui peuvent être levées par l'une de leurs opérations, par déclaration de surnom (par exemple, **Non_Existent: exception renames EXTRA_ACM_Exceptions.Non_Existent;**).

3.1.3.5 Initialisations

Les objets de type **Resource**, **Event**, **Buffer** et **Barrier** peuvent être *initialisés* lors d'un appel à une fonction **Create**. Ceci est indiqué par un paramètre **Parameters** de type **Initial_Parameters**.

Les types **Initial_Parameters** sont des articles avec un discriminant **Change** de type **Boolean**.

Pour chaque type **Initial_Parameters**, il existe deux constantes: **No_Change** et **Default**.

La constant **No_Change** correspond à une valeur du discriminant **Change** égale à **False**.

Lorsque le discriminant **Change** du paramètre **Parameters** a la valeur **False** (**No_Change**), un objet n'est pas initialisé, ou il est initialisé par défaut.

Lorsque le discriminant **Change** du paramètre **Parameters** a la valeur **True**, on distingue les cas suivants:

- pour un objet configuré statiquement, les paramètres d'initialisation doivent être exactement les mêmes que les caractéristiques de la configuration pour l'objet considéré, sinon l'exception **Inconsistent** est levée;
- pour un objet créé dynamiquement, lorsque ce n'est pas le premier appel à la fonction **Create** pour l'objet considéré, les paramètres d'initialisation doivent être exactement les mêmes que les caractéristiques de l'objet considéré, sinon l'exception **Inconsistent** est levée;
- pour le premier appel à une fonction **Create** pour un objet créé dynamiquement, l'objet nouvellement créé prend comme caractéristiques initiales les valeurs des paramètres d'initialisation.

Pour un objet créé dynamiquement, les caractéristiques initiales d'un objet nouvellement créé par un premier appel à une fonction **Create** avec la valeur **False** comme valeur du discriminant du paramètre **Parameters** prennent des valeurs par défaut.

3.2 Resources

3.2.1 Introduction

1 A *resource* is a synchronization object allowing control of access to software or hardware resources.

2 Conceptually, a limited number of tasks (defined by the characteristics of the resource) may *seize* a resource simultaneously, that is, these tasks will have made a *seizure* of the resource. A resource is said to be *busy* when no further task may request a seizure. A task may be blocked when trying to seize a busy resource. A task *releases* a resource to allow the possibility of a new seizure. Each resource is initialized with the following characteristics:

- 3 - *capacity* is the maximum number of possible successive new seizures without any release,
- 4 - *initial* is the initial number of possible successive new seizures without any release.

5 Conceptually, a resource is composed of:

- 6 - an integer counter (initially non-negative),
- 7 - a queue of waiting tasks (initially empty) called the *waiting queue*.

8 An implementation may set limits on the sizes of waiting queues for resources.

9 A resource always satisfies the following invariants:

- 10 - counter ≥ 0 implies that the waiting queue is empty
- 11 - counter ≤ 0 implies that the length of the waiting queue is equal to the absolute value of the counter
- 12 - counter = (initial value
- current number of tasks in the waiting queue
- number of seizures
+ number of releases).

13 Note: Resources should not be used to share data: their operations are not Ada synchronization points, so optimizers may break the desired protections.

3.2.2 Resource management

1 The package `EXTRA_Resources` defines the private type `Resource` as follows:

2 **type Resource is private;**

3 In the remainder of this clause, the word *resource* is used to designate an object of the type `Resource`.

4 A constant `Null_Accessor` is declared. It indicates a nonexistent resource.

5 **Null_Accessor: constant Resource;**

6 By default, a variable of type `Resource` is initialized to `Null_Accessor`.

7 Once created (`Create`), a resource may be seized (`Get`) and released (`Release`); it may be observed (`Capacity_Of`, `Count_Of`, `Value_Of`, `Name_Of`) and destroyed (`Destroy`).

8 The type `Resource` is an accessor type which depends on the implementation.

9 All of the operations defined on `Resource`, except `Create` and `Destroy`, raise the exception `Non_Existing` if the parameter of type `Resource` does not identify an existing resource.

10 An implementation may set limits on the number of simultaneous existing resources. This shall be documented. When such limits exist, the exception `Limit_Exceeded` is raised if an attempt is made to create a new resource and that creation would exceed a limit.

11 The subprograms `Get` of package `EXTRA_Resources` are potentially blocking operations. In addition, all of the other subprograms are potentially blocking operations if they apply to a distributed object; otherwise, they are not potentially blocking operations.

3.2 Ressources

3.2.1 Introduction

Une *ressource* est un objet de synchronisation permettant de réaliser le contrôle d'accès à des ressources logicielles ou matérielles.

Conceptuellement, un nombre limité (défini par les caractéristiques de la ressource) de tâches peuvent *tenir* simultanément une ressource: c'est à dire que ces tâches auront effectué une *prise* de la ressource. Une ressource est dite *occupée* lorsqu'aucune tâche ne peut faire de nouvelle prise. Une tâche peut être bloquée lors d'une tentative de prise d'une ressource occupée. Une tâche *libère* une ressource pour permettre une nouvelle possibilité de prise. Chaque ressource est initialisée avec les caractéristiques suivantes:

- *capacité* est le nombre maximum de possibilités de prises nouvelles sans qu'il y ait besoin de libération;
- *initial* est le nombre initial de possibilités de prises nouvelles sans qu'il y ait besoin de libération.

Conceptuellement, une ressource est composée:

- d'un compteur entier (initialement non négatif),
- d'une file de tâches en attente de prise (initialement vide) appelée par la suite *file d'attente*.

Une implémentation peut fixer des limites aux tailles des files d'attente des ressources.

Une ressource satisfait toujours les invariants suivants:

- compteur ≥ 0 implique que la file d'attente est vide,
- compteur ≤ 0 implique que la longueur de la file d'attente est égale à la valeur absolue du compteur,
- compteur = (valeur initiale
- nombre courant de tâches dans la file d'attente
- nombre de prises
+ nombre de libérations)

Note: Il est recommandé de ne pas utiliser les ressources pour le partage de données: leurs opérations ne sont pas des points de synchronisation Ada et ainsi les optimiseurs peuvent casser les protections souhaitées.

3.2.2 Gestion de ressource

Le paquetage EXTRA_Resources définit le type privé Resource comme suit:

```
type Resource is private;
```

Dans le reste du présent article, le terme *ressource* sera utilisé pour désigner un objet de type Resource.

Une constante Null_Accessor est fournie. Elle correspond à une ressource non existante.

```
Null_Accessor: constant Resource;
```

Par défaut, une variable de type Resource est initialisée à Null_Accessor.

Une fois créée (Create) une ressource peut être prise (Get) et libérée (Release); elle peut être consultée (Capacity_Of, Count_Of, Value_Of, Name_Of) et détruite (Destroy).

Le type Resource est un type désignant qui dépend de l'implémentation.

Toutes les opérations définies sur Resource, sauf Create, et Destroy, lèvent l'exception Non_Existant si le paramètre de type Resource n'identifie pas une ressource existante.

Une implémentation peut fixer des limites quant au nombre des ressources existant simultanément. Cela doit être documenté. Lorsque de telles limites existent, l'exception Limit_Exceeded est levée lors d'une tentative de création d'une nouvelle ressource qui ferait qu'une limite serait dépassée.

Les sous-programmes Get du paquetage EXTRA_Resources sont des opérations potentiellement bloquantes. De plus, tous les autres sous-programmes sont des opérations potentiellement bloquantes si elles s'appliquent à un objet réparti, sinon elles ne sont pas des opérations potentiellement bloquantes.

3.2.3 Specification of the package EXTRA_Resources

```

1 with EXTRA_ACM_Exceptions;
package EXTRA_Resources is

    type Resource is private;
    Null_Accessor      : constant Resource;
    Immediately        : constant Duration := 0.0;
    Anonymous          : constant String := "";
    Max_Capacity       : constant := <positive-integer-implementation-defined>;
    Max_Waiting_Tasks  : constant := <positive-integer-implementation-defined>;

    type Initial_Range is range 0 .. Max_Capacity;
    subtype Capacity_Range is Initial_Range range 1 .. Initial_Range'Last;
    type Counter_Range is range - Max_Waiting_Tasks .. Max_Capacity;
    type Waiting_Range is range 0 .. Max_Waiting_Tasks;

    type Initial_Parameters (Change : Boolean) is record
        case Change is
            when True =>      Initial:      Initial_Range;
                           Capacity:    Capacity_Range;
            when False =>     null;
        end case;
    end record;
    No_Change      : constant Initial_Parameters := (Change => False);
    Default        : constant Initial_Parameters := (Change => True,
                                                    Initial => 1, Capacity => 1);

    function Create      (Name           : in String := Anonymous;
                          Parameters     : in Initial_Parameters := Default)
                          return Resource;

    procedure Get        (A_Resource     : in Resource);
    procedure Get        (A_Resource     : in Resource;
                          Passed         : out Boolean;
                          Time_Out       : in Duration := Immediately);

    procedure Release    (A_Resource     : in Resource);
    procedure Destroy    (A_Resource     : in out Resource);
    function Capacity_Of (A_Resource     : in Resource) return Capacity_Range;
    function Count_Of    (A_Resource     : in Resource) return Waiting_Range;
    function Value_Of    (A_Resource     : in Resource) return Counter_Range;
    function Name_Of     (A_Resource     : in Resource) return String;

    Non_Existent         : exception renames EXTRA_ACM_Exceptions.Non_Existent;
    Inconsistent         : exception renames EXTRA_ACM_Exceptions.Inconsistent;
    Invalid_Name         : exception renames EXTRA_ACM_Exceptions.Invalid_Name;
    Unreachable          : exception renames EXTRA_ACM_Exceptions.Unreachable;
    Waiting_Task_Overflow : exception renames
                                EXTRA_ACM_Exceptions.Waiting_Task_Overflow;
    Limit_Exceeded       : exception renames EXTRA_ACM_Exceptions.Limit_Exceeded;
    Capacity_Overflow     : exception;

private
    -- not specified by this Technical Report
end EXTRA_Resources;

```

3.2.3 Spécification du paquetage EXTRA_Resources

1

```

with EXTRA_ACM_Exceptions;
package EXTRA_Resources is

  type Resource is private;
  Null_Accessor      : constant Resource;
  Immediately        : constant Duration := 0.0;
  Anonymous          : constant String := "";
  Max_Capacity       : constant := <positive-integer-implementation-defined>;
  Max_Waiting_Tasks  : constant := <positive-integer-implementation-defined>;

  type Initial_Range is range 0 .. Max_Capacity;
  subtype Capacity_Range is Initial_Range range 1 .. Initial_Range'Last;
  type Counter_Range is range - Max_Waiting_Tasks .. Max_Capacity;
  type Waiting_Range is range 0 .. Max_Waiting_Tasks;

  type Initial_Parameters (Change : Boolean) is record
    case Change is
      when True =>      Initial:   Initial_Range;
                       Capacity:  Capacity_Range;
      when False =>     null;
    end case;
  end record;
  No_Change      : constant Initial_Parameters := (Change => False);
  Default        : constant Initial_Parameters := (Change => True,
                                                    Initial => 1, Capacity => 1);

  function Create      (Name           : in String := Anonymous;
                        Parameters      : in Initial_Parameters := Default)
                        return Resource;

  procedure Get        (A_Resource     : in Resource);
  procedure Get        (A_Resource     : in Resource;
                        Passed          : out Boolean;
                        Time_Out        : in Duration := Immediately);

  procedure Release    (A_Resource     : in Resource);
  procedure Destroy    (A_Resource     : in out Resource);
  function Capacity_Of (A_Resource     : in Resource) return Capacity_Range;
  function Count_Of    (A_Resource     : in Resource) return Waiting_Range;
  function Value_Of    (A_Resource     : in Resource) return Counter_Range;
  function Name_Of     (A_Resource     : in Resource) return String;

  Non_Existent        : exception renames EXTRA_ACM_Exceptions.Non_Existent;
  Inconsistent        : exception renames EXTRA_ACM_Exceptions.Inconsistent;
  Invalid_Name        : exception renames EXTRA_ACM_Exceptions.Invalid_Name;
  Unreachable         : exception renames EXTRA_ACM_Exceptions.Unreachable;
  Waiting_Task_Overflow : exception
                        renames EXTRA_ACM_Exceptions.Waiting_Task_Overflow;
  Limit_Exceeded      : exception renames EXTRA_ACM_Exceptions.Limit_Exceeded;
  Capacity_Overflow   : exception;

private
  -- non spécifié par le présent Rapport Technique
end EXTRA_Resources;

```

3.2.4 Semantic description

- 1 **function** Create (Name : **in** String := Anonymous;
 Parameters : **in** Initial_Parameters := Default)
 return Resource;
- 2 Creates a logical object and returns an accessor to a resource either local or distributed, depending on the value of Name. If the discriminant Change of the parameter Parameters has the value True, the components Initial and Capacity of this same parameter give the characteristics initial and capacity of the resource. At the first call to the function Create with a given name, if the discriminant Change of the parameter Parameters has the value False, the characteristics initial and capacity of the resource are either fixed by the configuration (static resource) or take the default values 1 for initial and 1 for capacity. When it is not the first call, neither the counter nor the waiting queue of the resource identified by parameter Name are modified.
- 3 The exception Invalid_Name is raised if the parameter Name does not identify a resource known to the implementation. The exception Inconsistent is raised if the value of the discriminant Change of the parameter Parameters is True and the components Initial and Capacity of this same parameter are not compatible with the identified object. The exception Limit_Exceeded is raised if some implementation-defined limit would be exceeded by the completion of this subprogram. The exception Unreachable is raised if communication between the caller and the resource is not possible for the time being. In this case, the state of the resource is undefined.
- 4 **procedure** Get (A_Resource : **in** Resource);
- 5 Unconditionally seizes a resource and decrements the counter of the resource by one unit. If the resource was not already busy (if its counter is positive or equal to zero after being decremented), the caller continues its execution without explicitly requesting any task dispatching action. Otherwise, the caller is blocked and added to the waiting queue. Task dispatching occurs. The caller may become ready by a call made by another task to procedure Release on the same resource.
- 6 The exception Non_Existing is raised if the parameter A_Resource has the value Null_Accessor or does not identify an existing resource. The exception Waiting_Task_Overflow is raised if there is an attempt to exceed the maximum size of the waiting queue. The exception Unreachable is raised if communication between the caller and the resource is not possible for the time being. In this case, the state of the resource is undefined.
- 7 **procedure** Get (A_Resource : **in** Resource;
 Passed : **out** Boolean;
 Time_Out : **in** Duration := Immediately);
- 8 Similar to the preceding procedure, but a «time-out» is added. If the seizure of the resource is possible before the duration given by parameter Time_Out has elapsed, then the value of Passed is set to the value True. Otherwise, the counter of the resource is incremented by one unit and the value of Passed is set to the value False and the seizure of the resource is abandoned. The value Immediately or a negative value for the parameter Time_Out guarantees the absence of waiting in the case when it is not possible to immediately seize the resource.
- 9 **procedure** Release (A_Resource : **in** Resource);
- 10 Releases a resource. Increments the counter by one unit. If, after incrementing, the value of the counter is negative or equal to zero, the task at the head of the waiting queue is removed from the waiting queue and becomes ready.
- 11 The exception Non_Existing is raised if the parameter A_Resource has the value Null_Accessor or does not identify an existing resource. The exception Capacity_Overflow is raised if the counter of the resource had already reached the capacity of the resource. The exception Unreachable is raised if communication between the caller and the resource is not possible for the time being. In this case, the state of the resource is undefined.

3.2.4 Description sémantique

```

function Create      (Name      : in String := Anonymous;
                      Parameters : in Initial_Parameters := Default)
return Resource;

```

Crée un objet logique et fournit en retour un désignant pour une ressource locale ou répartie, selon la valeur de Name. Si le discriminant Change du paramètre Parameters a la valeur True, les composants Initial et Capacity de ce même paramètre fournissent les caractéristiques «initial» et «capacité» de la ressource. Pour le premier appel à la fonction Create avec un nom donné, si le discriminant Change du paramètre Parameters a la valeur False, les caractéristiques «initial» et «capacité» de la ressource soit sont fixées par la configuration (ressource statique) soit prennent les valeurs par défaut 1 pour «initial» et 1 pour «capacité». Quand ce n'est pas le premier appel, ni le compteur ni la file d'attente de la ressource identifiée par le paramètre Name ne sont modifiés.

L'exception Invalid_Name est levée si le paramètre Name n'identifie pas une ressource connue de l'implémentation. L'exception Inconsistent est levée quand la valeur du discriminant Change du paramètre Parameters est True et que les composants Initial et Capacity de ce même paramètre ne sont pas compatibles avec l'objet identifié. L'exception Limit_Exceeded est levée si une limite fixée par l'implémentation serait dépassée par la réussite de ce sous-programme. L'exception Unreachable est levée si la communication entre l'appelant et la ressource est impossible temporairement. Dans ce cas, l'état de la ressource est indéfini.

```

procedure Get (A_Resource : in Resource);

```

Prend inconditionnellement une ressource et décrémente le compteur de la ressource d'une unité. Si la ressource n'était pas déjà occupée (si son compteur est positif ou nul à la suite de la décrémentation), l'appelant continue son exécution sans provoquer d'action spécifique d'ordonnancement des tâches. Sinon, l'appelant est bloqué et rajouté à la file d'attente; il y a un ordonnancement des tâches; l'appelant peut devenir prêt par un appel à la procédure Release sur la même ressource faite par une autre tâche.

L'exception Non_Existente est levée si le paramètre A_Resource a la valeur Null_Accessor ou n'identifie pas une ressource existante. L'exception Waiting_Task_Overflow est levée s'il y a tentative de dépassement de la taille maximum de la file d'attente. L'exception Unreachable est levée si la communication entre l'appelant et la ressource est impossible temporairement. Dans ce cas, l'état de la ressource est indéfini.

```

procedure Get      (A_Resource : in Resource;
                    Passed      : out Boolean;
                    Time_Out    : in Duration := Immediately);

```

Similaire à la procédure précédente, mais un «time-out» est ajouté: si la prise de la ressource est possible avant l'écoulement de la durée donnée dans le paramètre Time_Out, alors la valeur de Passed est positionnée à la valeur True. Sinon, le compteur de la ressource est incrémenté d'une unité, la valeur de Passed est positionnée à la valeur False et la prise de la ressource est abandonnée. La valeur Immediately ou une valeur négative pour le paramètre Time_Out garantit l'absence d'attente en cas d'impossibilité de prise immédiate de la ressource.

```

procedure Release (A_Resource : in Resource);

```

Libère une ressource. Incrémente le compteur d'une unité. Si, après incrémentation, la valeur du compteur est négative ou nulle, la tâche en tête de la file d'attente est enlevée de la file et devient prête à nouveau.

L'exception Non_Existente est levée si le paramètre A_Resource a la valeur Null_Accessor ou n'identifie pas une ressource existante. L'exception Capacity_Overflow est levée si le compteur de la ressource avait déjà atteint la capacité de la ressource. L'exception Unreachable est levée si la communication entre l'appelant et la ressource est impossible temporairement. Dans ce cas, l'état de la ressource est indéfini.

12 **procedure** Destroy (A_Resource : in out Resource);

13 Destroys the resource identified by the parameter A_Resource. The resource identified by parameter A_Resource is no longer accessible. If the waiting queue is not empty, all waiting tasks become ready and the exception Non_Existing is propagated to them. The value of A_Resource is set to Null_Accessor. If the parameter A_Resource does not identify an existing resource, this procedure has no other effect than to set the value of A_Resource to Null_Accessor.

14 The exception Unreachable is raised if communication between the caller and the resource is not possible for the time being. In this case, the state of the resource is undefined.

15 **function** Capacity_Of (A_Resource : in Resource) return Capacity_Range;

16 Returns the capacity of a resource.

17 The exception Non_Existing is raised if the parameter A_Resource has the value Null_Accessor or does not identify an existing resource. The exception Unreachable is raised if communication between the caller and the resource is not possible for the time being.

18 **function** Count_Of (A_Resource : in Resource) return Waiting_Range;

19 Returns the current number of tasks in the waiting queue associated with a resource.

20 Note: Users of this function should take the same precautions they take when using an attribute 'Count on a task entry.

21 The exception Non_Existing is raised if the parameter A_Resource has the value Null_Accessor or does not identify an existing resource. The exception Unreachable is raised if communication between the caller and the resource is not possible for the time being.

22 **function** Value_Of (A_Resource : in Resource) return Counter_Range;

23 Returns the value of the counter of a resource (if positive, it is the number of possible new seizures; if negative, it is the opposite of the current number of tasks in the waiting queue of the resource).

24 Note: Users of this function should take the same precautions they take when using an attribute 'Count on a task entry.

25 The exception Non_Existing is raised if the parameter A_Resource has the value Null_Accessor or does not identify an existing resource. The exception Unreachable is raised if communication between the caller and the resource is not possible for the time being.

26 **function** Name_Of (A_Resource : in Resource) return String;

27 Returns the name of a resource. This name is equivalent to the parameter Name of the function Create with which the accessor was obtained.

28 The exception Non_Existing is raised if the parameter A_Resource has the value Null_Accessor or does not identify an existing resource. The exception Unreachable is raised if communication between the caller and the resource is not possible for the time being.

procedure Destroy (A_Resource : in out Resource);	12
Détruit la ressource identifiée par le paramètre A_Resource. La ressource identifiée par le paramètre A_Resource n'est plus accessible. Si la file d'attente n'est pas vide, toutes les tâches en attente deviennent prêtes à et l'exception Non_Existent leur est propagée. La valeur de A_Resource est positionnée à Null_Accessor. Si le paramètre A_Resource n'identifie pas une ressource existante, cette procédure a pour seul effet de positionner la valeur de A_Resource à Null_Accessor.	13
L'exception Unreachable est levée si la communication entre l'appelant et la ressource est impossible temporairement. Dans ce cas, l'état de la ressource est indéfini.	14
function Capacity_Of (A_Resource : in Resource) return Capacity_Range;	15
Fournit en retour la capacité d'une ressource .	16
L'exception Non_Existent est levée si le paramètre A_Resource a la valeur Null_Accessor ou n'identifie pas une ressource existante. L'exception Unreachable est levée si la communication entre l'appelant et la ressource est impossible temporairement.	17
function Count_Of (A_Resource : in Resource) return Waiting_Range;	18
Fournit en retour le nombre courant des tâches de la file d'attente associée à une ressource.	19
Note: Les utilisateurs de cette fonction doivent prendre les mêmes précautions que celles qu'ils prennent lorsqu'ils utilisent un attribut 'Count d'une entrée de tâche.	20
L'exception Non_Existent est levée si le paramètre A_Resource a la valeur Null_Accessor ou n'identifie pas une ressource existante. L'exception Unreachable est levée si la communication entre l'appelant et la ressource est impossible temporairement.	21
function Value_Of (A_Resource : in Resource) return Counter_Range;	22
Fournit en retour la valeur du compteur d'une ressource (si elle est positive, c'est le nombre de nouvelles prises possibles; si elle est négative, c'est l'opposé du nombre courant des tâches de la file d'attente de la ressource).	23
Note: Les utilisateurs de cette fonction doivent prendre les mêmes précautions que celles qu'ils prennent lorsqu'ils utilisent un attribut 'Count d'une entrée de tâche.	24
L'exception Non_Existent est levée si le paramètre A_Resource a la valeur Null_Accessor ou n'identifie pas une ressource existante. L'exception Unreachable est levée si la communication entre l'appelant et la ressource est impossible temporairement.	25
function Name_Of (A_Resource : in Resource) return String;	26
Fournit en retour le nom d'une ressource. Ce nom est équivalent au paramètre Name avec lequel le désignant a été obtenu.	27
L'exception Non_Existent est levée si le paramètre A_Resource a la valeur Null_Accessor ou n'identifie pas une ressource existante. L'exception Unreachable est levée si la communication entre l'appelant et la ressource est impossible temporairement.	28

3.3 Events

3.3.1 Introduction

- 1 An event is a synchronization object to indicate the state of a condition. There are two possible states: *up*, indicating that the condition is true, and *down*, indicating it is false.
- 2 A task may change the state of an event. A task may wait until the state of an event takes a specified value (up or down)
- 3 Conceptually, an event is composed of :
 - 4 - a bivalued state variable (states up and down),
 - 5 - a set of tasks waiting for a specified value of the state (initially empty) called *waiting set*.
- 6 Note: For a given event, when the state of the event is up, the tasks of the waiting set wait for the state to become down, while when the state of the event is down, the tasks of the waiting set wait for the state to become up.

3.3.2 Event management

- 1 The package `EXTRA_Events` defines the private type `Event` as follows:
 - 2 `type Event is private;`
- 3 In the remainder of this clause, the word *event* is used to designate an object of the type `Event`.
- 4 A constant `Null_Accessor` is declared. It indicates a nonexistent event.
 - 5 `Null_Accessor: constant Event;`
- 6 By default, a variable of type `Event` is initialized to `Null_Accessor`.
- 7 Once an event is created (`Create`), its state may set to up (`Set`), to down (`Reset`) or may be changed (`Toggle`). Tasks may wait until its state takes a specified value (`Wait`). An event may be observed (`Count_Of`, `State_Of`, `Name_Of`) and destroyed (`Destroy`).
- 8 The `Event` type is an accessor type which depends on the implementation.
- 9 All of the operations defined on `Event`, except `Create` and `Destroy`, raise the exception `Non_Existent` if the parameter of type `Event` does not identify an existing event.
- 10 An implementation may set limits on the number of simultaneous existing events. This shall be documented. When such limits exist, the exception `Limit_Exceeded` is raised if an attempt is made to create a new event and that creation would exceed a limit.
- 11 The subprograms `Wait` of package `EXTRA_Events` are potentially blocking operations. In addition, all of the other subprograms are potentially blocking operations if they apply to a distributed object ; otherwise, they are not potentially blocking operations.

3.3 Événements à niveau

3.3.1 Introduction

Un événement à niveau est un objet de synchronisation pour indiquer l'état d'une condition. Il a deux états possibles: état *haut* indiquant que la condition est vraie, et état *bas* indiquant que la condition est fausse .

Une tâche peut changer l'état d'un événement à niveau. Une tâche peut attendre jusqu'à ce que l'état d'un événement à niveau prenne une valeur spécifiée (haut ou bas).

Conceptuellement, un événement à niveau est composé de:

- une variable bivaluée d'état (états haut et bas) ,
- un ensemble de tâches en attente d'une valeur spécifiée de l'état (initialement vide) appelé par la suite *ensemble d'attente* .

Note: Pour un événement à niveau donné, lorsque l'état de l'événement est haut, les tâches de l'ensemble d'attente attendent le passage de l'état à l'état bas, tandis que lorsque l'état de l'événement est bas, les tâches de l'ensemble d'attente attendent le passage de l'état à l'état haut.

3.3.2 Gestion d'événements

Le paquetage `EXTRA_Events` définit le type privé `Event` comme suit:

```
type Event is private;
```

Dans le reste du présent article, le terme *événement à niveau* sera utilisé pour désigner un objet de type `Event`.

Une constante `Null_Accessor` est fournie. Elle correspond à un événement à niveau non existant.

```
Null_Accessor: constant Event;
```

Par défaut, une variable de type `Event` est initialisée à `Null_Accessor`.

Une fois un événement à niveau créé (`Create`), son état peut être positionné haut (`Set`) ou bas (`Reset`) ou changé (`Toggle`). Les tâches peuvent attendre que l'état prenne une valeur spécifiée (`Wait`). Un événement à niveau peut être consulté (`Count_Of`, `State_Of`, `Name_Of`) et détruit (`Destroy`).

Le type `Event` est un type désignant qui dépend de l'implémentation.

Toutes les opérations définies sur `Event`, sauf `Create` et `Destroy`, lèvent l'exception `Non_Existent` si le paramètre de type `Event` n'identifie pas un événement à niveau existant.

Une implémentation peut fixer des limites quant au nombre des événements à niveau existant simultanément. Cela doit être documenté. Lorsque de telles limites existent, l'exception `Limit_Exceeded` est levée lors d'une tentative de création d'un nouvel événement à niveau qui ferait qu'une limite serait dépassée.

Les sous-programmes `Wait` du paquetage `EXTRA_Events` sont des opérations potentiellement bloquantes. De plus, tous les autres sous-programmes sont des opérations potentiellement bloquantes si elles s'appliquent à un objet réparti, sinon elles ne sont pas des opérations potentiellement bloquantes.

3.3.3 Specification of the package EXTRA_Events

```

1 with EXTRA_ACM_Exceptions;
package EXTRA_Events is

    type Event is private;
    Null_Accessor      : constant Event;
    Immediately        : constant Duration := 0.0;
    Anonymous          : constant String := "";
    Max_Waiting_Tasks  : constant := <positive-integer-implementation-defined>;

    type Waiting_Range is range 0 .. Max_Waiting_Tasks;

    type State is (Up, Down);

    type Initial_Parameters (Change : Boolean) is record
        case Change is
            when True => Initial: State;
            when False => null;
        end case;
    end record;
    No_Change      : constant Initial_Parameters := (Change => False);
    Default        : constant Initial_Parameters := (Change => True,
                                                    Initial => Down);

    function Create      (Name          : in String := Anonymous;
                          Parameters    : in Initial_Parameters := Default)
                          return Event;

    procedure Wait       (An_Event      : in Event;
                          A_State       : in State := Up);

    procedure Wait       (An_Event      : in Event;
                          Passed        : out Boolean;
                          Time_Out      : in Duration := Immediately;
                          A_State       : in State := Up);

    procedure Set        (An_Event      : in Event);
    procedure Reset      (An_Event      : in Event);
    procedure Toggle     (An_Event      : in Event);
    procedure Destroy    (An_Event      : in out Event);
    function Count_Of    (An_Event      : in Event) return Waiting_Range;
    function State_Of    (An_Event      : in Event) return State;
    function Name_Of     (An_Event      : in Event) return String;

    -- Exceptions
    Inconsistent          : exception renames EXTRA_ACM_Exceptions.Inconsistent;
    Non_Existnt          : exception renames EXTRA_ACM_Exceptions.Non_Existnt;
    Invalid_Name         : exception renames EXTRA_ACM_Exceptions.Invalid_Name;
    Unreachable          : exception renames EXTRA_ACM_Exceptions.Unreachable;
    Waiting_Task_Overflow : exception
renames EXTRA_ACM_Exceptions.Waiting_Task_Overflow;
    Limit_Exceeded       : exception renames EXTRA_ACM_Exceptions.Limit_Exceeded;
private
    -- not specified by this Technical Report
end EXTRA_Events;

```

3.3.3 Spécification du paquetage EXTRA_Events

1

```

with EXTRA_ACM_Exceptions;
package EXTRA_Events is

  type Event is private;
  Null_Accessor      : constant Event;
  Immediately        : constant Duration := 0.0;
  Anonymous          : constant String := "";
  Max_Waiting_Tasks  : constant := <positive-integer-implementation-defined>;

  type Waiting_Range is range 0 .. Max_Waiting_Tasks;

  type State is (Up, Down);

  type Initial_Parameters (Change : Boolean) is record
    case Change is
      when True => Initial: State;
      when False => null;
    end case;
  end record;
  No_Change      : constant Initial_Parameters := (Change => False);
  Default        : constant Initial_Parameters := (Change => True,
                                                    Initial => Down);

  function Create      (Name          : in String := Anonymous;
                        Parameters    : in Initial_Parameters := Default)
                        return Event;

  procedure Wait       (An_Event      : in Event;
                        A_State        : in State := Up);

  procedure Wait       (An_Event      : in Event;
                        Passed         : out Boolean;
                        Time_Out       : in Duration := Immediately;
                        A_State        : in State := Up);

  procedure Set        (An_Event      : in Event);
  procedure Reset      (An_Event      : in Event);
  procedure Toggle     (An_Event      : in Event);
  procedure Destroy    (An_Event      : in out Event);
  function Count_Of    (An_Event      : in Event) return Waiting_Range;
  function State_Of    (An_Event      : in Event) return State;
  function Name_Of     (An_Event      : in Event) return String;

  -- Exceptions
  Inconsistent         : exception renames EXTRA_ACM_Exceptions.Inconsistent;
  Non_Existent         : exception renames EXTRA_ACM_Exceptions.Non_Existent;
  Invalid_Name         : exception renames EXTRA_ACM_Exceptions.Invalid_Name;
  Unreachable          : exception renames EXTRA_ACM_Exceptions.Unreachable;
  Waiting_Task_Overflow : exception
                        renames EXTRA_ACM_Exceptions.Waiting_Task_Overflow;
  Limit_Exceeded       : exception renames EXTRA_ACM_Exceptions.Limit_Exceeded;
private
  -- non spécifié par le présent Rapport Technique
end EXTRA_Events;

```

3.3.4 Semantic description

- 1 **function** Create (Name : **in** String := Anonymous;
Parameters : **in** Initial_Parameters := Default)
 return Event;
- 2 Creates a logical object and returns an accessor to an event either local or distributed, depending on the value of Name. If the discriminant Change of the parameter Parameters has the value True, the component Initial of this same parameter gives the initial value of the state of the event. At the first call to the function Create with a given name, if the discriminant Change of the parameter Parameters has the value False, the initial value of the state of the event either is fixed by the configuration (static event) or takes the default value Down. When it is not the first call, neither the state nor the waiting set of the event identified by parameter Name is modified.
- 3 The exception Invalid_Name is raised if the parameter Name does not identify an event known to the implementation. The exception Inconsistent is raised if the value of the discriminant Change of the parameter Parameters is True and the component Initial of this same parameter is not compatible with the identified object. The exception Limit_Exceeded is raised if some implementation-defined limit would be exceeded by the completion of this subprogram. The exception Unreachable is raised if communication between the caller and the event is not possible for the time being. In this case, the state of the event is undefined.
- 4 **procedure** Wait (An_Event : **in** Event;
 A_State : **in** State := Up);
- 5 Unconditionally waits on an event. If the state of the event is equal to the value of A_State, the caller continues its execution without explicitly requesting any task dispatching action. Otherwise, the caller is blocked and added to the waiting set. Task dispatching occurs. The caller may become ready by a call made by another task to a procedure Set, Reset or Toggle on the same event which changes the state of the event.
- 6 The exception Non_Existing is raised if the parameter An_Event has the value Null_Accessor or does not identify an existing event. The exception Waiting_Task_Overflow is raised if there is an attempt to exceed the maximum size of the waiting set. The exception Unreachable is raised if communication between the caller and the event is not possible for the time being. In this case, the state of the event is undefined.
- 7 **procedure** Wait (An_Event : **in** Event;
 Passed : **out** Boolean;
 Time_Out : **in** Duration := Immediately;
 A_State : **in** State := Up);
- 8 Similar to the preceding procedure, but a «time-out» is added. If the state of the event is equal to the value of A_State before the duration given by parameter Time_Out has elapsed, then the value of Passed is set to the value True. Otherwise, the value of Passed is set to the value False and the waiting is abandoned. The value Immediately or a negative value for the parameter Time_Out guarantees the absence of waiting in the case when the state of the event is equal to the value of A_State immediately prior to invocation of this procedure.
- 9 **procedure** Set (An_Event : **in** Event);
- 10 Sets an event to the state up. If the state of the event was down immediately prior to invocation of this procedure and the waiting set is not empty, all of the tasks of the waiting set are removed from the waiting set and become ready. Otherwise, the caller continues its execution without explicitly requesting any task dispatching action.
- 11 The exception Non_Existing is raised if the parameter An_Event has the value Null_Accessor or does not identify an existing event. The exception Unreachable is raised if communication between the caller and the event is not possible for the time being. In this case, the state of the event is undefined.

3.3.4 Description sémantique

```

function Create      (Name      : in String := Anonymous;
                      Parameters : in Initial_Parameters := Default)
                      return Event;

```

Crée un objet logique et fournit en retour un désignant pour un événement à niveau soit local, soit réparti, selon la valeur de Name . Si le discriminant Change du paramètre Parameters a la valeur True, le composant Initial de ce même paramètre fournit la valeur initiale de l'état de l'événement à niveau. Pour le premier appel à la fonction Create avec un nom donné, si le discriminant Change du paramètre Parameters a la valeur False, la valeur initiale de l'état de l'événement à niveau soit est fixée par la configuration (événement à niveau statique) soit prend la valeurs par défaut Down. Quand ce n'est pas le premier appel, ni l'état ni l'ensemble d'attente de l'événement à niveau identifié par le paramètre Name ne sont modifiés.

L'exception Invalid_Name est levée si le paramètre Name n'identifie pas un événement à niveau connu de l'implémentation. L'exception Inconsistent est levée si la valeur du discriminant Change du paramètre Parameters est True et que le composant Initial de ce même paramètre n'est pas compatible avec l'objet identifié. L'exception Limit_Exceeded est levée si une limite fixée par l'implémentation serait dépassée par la réussite de ce sous-programme. L'exception Unreachable est levée si la communication entre l'appelant et l'événement à niveau est impossible temporairement. Dans ce cas, l'état de l'événement à niveau est indéfini.

```

procedure Wait      (An_Event   : in Event
                     A_State     : in State := Up);

```

Attend inconditionnellement sur un événement à niveau. Si l'état de l'événement à niveau est égal à la valeur de A_State, l'appelant continue son exécution, sans provoquer d'action spécifique d'ordonnancement des tâches. Autrement, l'appelant est bloqué et rajouté à l'ensemble d'attente; il y a ordonnancement des tâches; l'appelant peut devenir prêt par un appel à une procédure Set, Reset ou Toggle sur cet événement à niveau fait par une autre tâche, qui change l'état de l'événement à niveau.

L'exception Non_Existent est levée si le paramètre An_Event a la valeur Null_Accessor ou n'identifie pas un événement à niveau existant. L'exception Waiting_Task_Overflow est levée s'il y a tentative de dépassement de la taille maximum de l'ensemble d'attente. L'exception Unreachable est levée si la communication entre l'appelant et l'événement à niveau est impossible temporairement. Dans ce cas, l'état de l'événement à niveau est indéfini.

```

procedure Wait      (An_Event   : in Event;
                     Passed      : out Boolean;
                     Time_Out    : in Duration := Immediately
                     A_State     : in State := Up);

```

Similaire à la procédure précédente, mais un «time-out» est ajouté: si l'état de l'événement à niveau est égal à la valeur de A_State avant l'écoulement de la durée donnée dans le paramètre Time_Out, alors la valeur de Passed est positionnée à la valeur True. Sinon, la valeur de Passed est positionnée à la valeur False et l'attente sur l'événement à niveau est abandonnée. La valeur Immediately ou une valeur négative pour le paramètre Time_Out garantit l'absence d'attente dans le cas où l'état de l'événement à niveau est égal à la valeur de A_State immédiatement avant l'invocation de cette procédure .

```

procedure Set (An_Event : in Event);

```

Positionne un événement à niveau à l'état haut. Si l'état de l'événement à niveau était bas immédiatement avant l'invocation de cette procédure et que l'ensemble d'attente n'est pas vide, toutes les tâches de cet ensemble sont supprimées de l'ensemble d'attente et deviennent prêtes. Sinon, l'appelant continue son exécution, sans provoquer d'action spécifique d'ordonnancement des tâches.

L'exception Non_Existent est levée si le paramètre An_Event a la valeur Null_Accessor ou n'identifie pas un événement à niveau existant. L'exception Unreachable est levée si la communication entre l'appelant et l'événement à niveau est impossible temporairement. Dans ce cas, l'état de l'événement à niveau est indéfini.