

Distributed Real-Time Specification for Java

Early Draft Review #1

Editor

Jonathan S. Anderson
Virginia Polytechnic and State University
The MITRE Corporation

E. Douglas Jensen The MITRE Corporation
Douglas M. Wells
Raymond K. Clark The MITRE Corporation
Yun Zhang The MITRE Corporation
Edward Burke The MITRE Corporation

Early Draft Review #1 Draft #1.3 — 16 April 2012

Copyright © 2001 – 2006 Jonathan Anderson
Copyright © 2001 – 2006 E. Douglas Jensen
Copyright © 2001 – 2006 The MITRE Corporation
Copyright © 2011 – 2012 aicas GmbH
All rights reserved

NOTICE

This technical data was produced for the U. S. Government under Contract No. FA8721-02-C-0001, and is subject to the Rights in Technical Data - Noncommercial Items clause at (DFARS) 252.227-7013 (NOV 1995)

© 2007 The MITRE Corporation. All Rights Reserved.

NOTICE

This software was produced for the U. S. Government under Contract No. FA8721-02-C-0001, and is subject to the Rights in Noncommercial Computer Software and Noncommercial Computer Software Documentation Clause (DFARS) 252.227-7014 (JUN 1995)

© 2007 The MITRE Corporation. All Rights Reserved.

No programmers were injured in the preparation of this document.

0.1.1

Publication History

EDR#1, Draft 1 (0.1.1)	14 September 2006
EDR#1, Draft 1 (0.1.2)	9 November 2007
EDR#1, Draft 1 (0.1.3)	16 April 2012

Typeset in Computer Modern using Peter R. Wilson's memoir class for L^AT_EX.

JSR-50 Expert Group Membership

James J. Hunt	aicas GmbH
Kelvin Nilsen	Atego Systems
Benjamin Brosgol	
E. Douglas Jensen	
Douglas M. Wells	
Raymond K. Clark	The MITRE Corporation
Andrew Wellings	University of York
Paul Lacrosse	Nortel
Esmond Pitt	
Gautam Thaker	

Specification and Implementation Team


Jonathan S. Anderson	The MITRE Corporation
Edward Burke	The MITRE Corporation
Ray Clark	The MITRE Corporation
Doug Wells	
Yun Zhang	The MITRE Corporation

Past Expert Group Members

Bill Beckwith OIS

External Contributors

Binoy Ravindran	Virginia Tech	Algorithm Guidance
Ed Curley	Virginia Tech	TMAR Protocols
Chewoo Na	Virginia Tech	Scheduler Implementations
Scott Robbins	The MITRE Corporation	Metascheduler



Contents

Contents	i
List of Figures	iii
1 Introduction	2
1.1 Requirements	2
1.2 Guiding Principles	2
1.3 Minimum Components for DRTSJ	5
1.4 Release Plan	6
2 Programming Model	7
2.1 Distributable Threads	7
Partial Failures	9
Implementation Implications	9
2.2 Scheduling	9
3 The Java and RTSJ Platforms	11
4 API Specification	12
4.1 Interfaces	12
DistributableThreadOperations	12
DistributedAsyncEventLocalMethods	14
DistributedAsynchronousEventHelper	14
DRTSJLocalServices	15
DThreadSegmentOperations	16
RealtimeRemote	19
RTRemoteAsynchronousEvent	19
RTRemoteAsynchronousEventHandler	20

4.2	Classes	20
	DistributableThread	20
	DistributedAsyncEvent	23
	DistributedAsyncEventHandler	24
	DistributedRealtimeClock	25
	DRTSJPlatform	25
	DRTSJServicesManager	32
	DThreadSegment	34
	GlobalIdentifier	35
	GlobalRealtimeThread	37
	GlobalThread	45
	RealtimeRemoteObject	52
4.3	Exceptions	54
	ThreadIntegrityViolation	54
5	Reference Implementation	56
	Realtime RMI	56
	Distributable Threads	57
	Thread Integrity	57
	Pluggable Scheduling	58
	The DRTSJ RI Distribution	59
	Demonstration Application	59
	Bibliography	61

List of Figures

2.1 Distributable Threads	8
-------------------------------------	---

Preface

The Distributed Real-Time Specification for Java (DRTSJ) is under development within the Java Community Process (JCP) by the membership of the JSR-50 Expert Group (EG). This group, was lead by the MITRE Corporation, but has been taken over by aicas GmbH. The group includes representation from individual, academic, U.S. government, defense, and industrial participants.[\[Dis\]](#)

Chapter 1

Introduction

The Real-Time Specification for Java (RTSJ) provides the necessary extensions to Java to enable programming realtime tasks in Java. This is a necessary prerequisite for distributed realtime programming, but not sufficient. The RTSJ does not provide any support for extending or coordinating computation across virtual machine boundaries, let alone machine boundaries. Standard Java provides Remote Message Invocation, but this mechanism is not sufficient for realtime programming. This specification provides the missing mechanism for cross machine realtime computation.

1.1 Requirements

The specification must fulfill two main goals. It should be compatible with other Java specifications that handle realtime behavior and it should extend that behavior beyond the virtual machine in a machine independent manner. Furthermore, the specification should leverage existing realtime distribution work from Realtime CORBA and other realtime distribution protocols without forcing the user to adapt to a programming style that is foreign to Java.

1.2 Guiding Principles

The EG recognizes the enormous variety of potential problem and solution spaces represented by the terms “distributed” and “realtime,” and the variety of opinions on what it could mean to construct distributed realtime

systems in the Java Programming Language. The JSR-50 proposal and the EG deliberately scoped the DRTSJ to a subset of those solution spaces and approaches to constructing distributed realtime systems, by extending the JSR-1 Real-Time Specification for Java (RTSJ) in a natural and familiar way, especially for Java programmers. We seek in this digest to briefly articulate this scope, summarize the work accomplished to date, and describe the intended products of the JSR-50 specification effort.

Several key assertions were debated and articulated by the EG in order to set the scope and define the relationship of JSR-50 to other work. Many of these deliberately follow the assertions made by the RTSJ EG, upon whose work much of the DRTSJ is predicated. [rts, JSRa, JSRb] Here we highlight some of these guiding principles, though we include by reference those called out by the RTSJ EG in [BBD⁺06, Introduction].

Bring distributed realtime to Java, not bring Java to traditional distributed realtime. [JJ, Jen02] It is the intention of the JSR-50 EG to bring facilities required for constructing the DRTSJ style of distributed realtime systems into the Java programming environment in a way that is least disruptive to experienced Java programmers. For instance, concurrent programming primitives such as the `synchronized` keyword and the `Thread` and related classes should remain familiar. Similarly, we build upon rather than replace Java's distributed object model as expressed in the Java RMI specification. A corollary is that the DRTSJ is, as its name indicates, specifically for Java, and is not intended to be language agnostic (as, for example, CORBA is more or less intended). The opposite approach of bringing Java to distributed realtime could conceivably be performed with a binding of the RTSJ to Real-Time CORBA, requiring Java programmers to learn something about how to use CORBA. Each approach is reasonable for some contexts, and our choice for the DRTSJ is not intended to compete with that alternative (should it eventually materialize), but instead to complement it and provide the users two options (at least).

Distributed objects and operations are distinguished from local objects and operations, for the obvious reasons of latency, partial failures, and concurrency control. [WWWK94]

Current Practice vs. Advanced Features: The DRTSJ will address current realtime system practice as well as allow future implementations to include advanced features. The EG has chosen to support this goal by

providing a specification which targets those technologies, techniques, and interfaces which have been well-tested, subjected to review, and successfully employed in the construction of non-trivial distributed realtime systems. Additional promising technologies under development by the research community which fail to meet those specification criteria may be provided in versions of the DRTSJ Reference Implementation (RI). Such versions of the RI serve as an active testing ground for innovative technologies which may appear in future versions of the specification.

Maintain the “flavor” of RTSJ: Java programmers who have already made the leap to the RTSJ should find adaptation to the DRTSJ as natural as possible. RTSJ applications should run unmodified on DRTSJ-compliant JVMs, with some caveats on their distributed behavior.

Do not dictate the use of an RTSJVM or realtime transport: The DRTSJ allows (the inevitable) mixtures of regular and RTSJ-compliant JVMs, and specifies the timeliness behaviors that result. It also is deliberately silent on the topic of the transport, which is regarded as a quality of implementation issue. As with Real-Time CORBA, realtime transports can be addressed in a subsequent specification. [Obj04]

Coherent support for end-to-end application properties: By definition, some multi-node behaviors in distributed realtime systems have end-to-end time constraints which must be respected for the system to perform acceptably. Other end-to-end properties may also be required, such as fault management, security credentials, serialization, etc. Traditionally these end-to-end properties have been forced on the application designers who must create bespoke and often ad-hoc mechanisms to attain them, typically at high recurring and non-recurring costs. The DRTSJ must provide basic facilities for passing and acting upon the required end-to-end context among the nodes participating in any given distributed behavior, in a manner that respects both the end-to-end argument [SRC84, Lam83] and the counter-examples, particularly those found in realtime systems. Conventional message-passing models (e.g., JMS [Sun02a]) and publish/subscribe models (e.g., OMGs DDS [Obj05]) tend to disregard the common need for multi-hop interactions, and hence make end-to-end properties the responsibility of the users.

Based on these driving premises, the keystone elements of the DRTSJ outlined in the introduction were selected. The distributable threads model,

discussed in detail in Section 2.1, provides a coherent, end-to-end abstraction for building concurrent, sequential activities for distributed systems in a manner familiar to Java and realtime programmers alike. In particular, the presence of a distributed object model (RMI) in the Java platform makes the distributable thread concept a straightforward step for Java programmers into the world of distributed realtime programming.

In non-trivial distributed systems, partial failures due to changing network conditions, node failures or overloads, and even regular maintenance must be considered as the common case rather than the exception. Therefore, any distributed system must provide facilities for detecting or masking these failures, and presenting the relevant events to the application. Again, the end-to-end argument and its counter-examples must drive engineering solutions in which responsibilities are assigned to different levels of any given system. The concept and implementations of distributable thread integrity, described in Section 2.1, are the primary means for meeting this requirement in the DRTSJ.

Finally, realtime — especially non-trivial distributed realtime — systems have a unique need for flexible, application-defined resource management. The RTSJ is extended in the DRTSJ by providing a scheduling framework. Software designers may provide user-level application-specific policies to govern the local and distributed scheduling of activities in the system. These policies may range from the RTSJ default of priorities, to deadline-based policies (not only the familiar “earliest deadline first,” but also ones widely used outside the classical hard realtime community, such as “minimize the number of missed deadlines,” “minimize mean tardiness,” etc.), to time/utility function utility accrual based policies [Rea].

1.3 Minimum Components for DRTSJ

The EG has settled on the following components as a minimum for the final specification:

Distributable Real-Time Threads, a proven programming model for constructing sequential control flow applications with end-to-end timeliness properties in distributed systems. The DRTSJ’s distributable threads are a realtime generalization of Java’s Remote Method Invocation, as originally proposed in JSR-50 (and are a superset of the abstraction provided in the OMG Real-Time CORBA specification 1.2 [Obj01];

A **Distributable Thread Integrity Framework**, into which application designers may plug appropriate policies for maintaining the health and integrity of distributable threads in the presence of failures; and

A **Scheduling Framework**, into which application designers may plug appropriate user space policies for scheduling distributable and local threads.

In the Fall of 2006, the EG will release an technology preview package called **Early Draft Review #1** (EDR#1) which will focus on the specification and implementation of the distributable threads abstraction. Subsequent Early Draft Reviews may be released to preview the distributable thread integrity framework, and the scheduling framework.

1.4 Release Plan

The EG expects to release a series of Early Draft Review (EDR) packages, of which this document is the first, to share our emerging vision of the DRTSJ with the JCP members and the wider community. We seek feedback from researchers and practitioners to leverage their perspectives on distributed realtime programming, and to ensure that emphasis is placed on problems for which there is a need for solutions.

Chapter 2

Programming Model

2.1 Distributable Threads

Many distributed systems have a natural expression as a collection of concurrent sequential flows of execution within and among objects. The *distributable thread* programming model (illustrated in Figure 2.1) supported in OMG’s Real-Time CORBA 1.2 standard (abbreviated here as RTC2) [OMG01] provides such threads as first-class abstractions. Distributable threads first appeared as “distributed threads” in the Alpha OS kernel [Nor87, JN90]. Subsequently the Alpha and Mach 3 microkernels were merged by the Open Software Foundation as the basis for its MK7.3 OS [Ope98], which (together with some other research OSs) substituted the (less accurate) term “migrating threads.”

A distributable thread is a single thread of execution with a globally unique identifier that extends and retracts through local and remote objects. Thus, a distributable thread is an end-to-end control flow abstraction, with a logically distinct locus of control flow movement within and among objects and nodes, that directly manifests the distributed behavior of many systems. This specification refer to distributable threads as *threads* except as necessary for clarity.

Threads extend and retract across object boundaries by performing remote procedure calls (RPCs). Therefore, a distributable thread may be seen as a chain of local threads (or *segments*) connected by intervening RPCs. A thread’s stack is thus distributed across the set of nodes hosting segments at any given time. While the synchrony of a conventional method

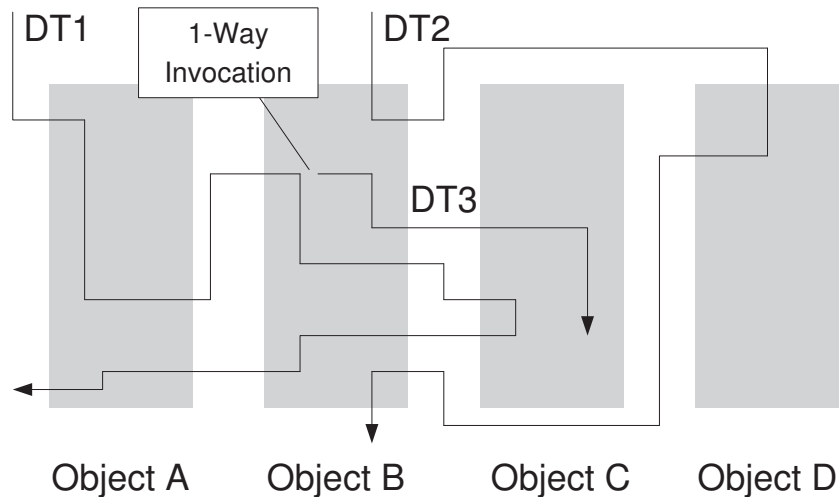


Figure 2.1: Distributable Threads

invocation is often cited as a concurrency limitation [SG01], a distributable thread is sequential rather than synchronous (“send/wait”). It is always executing somewhere (unless it is “in-flight” during an RPC communication step), while it is the most eligible there — it is not doing send/wait’s as with RPC. Each node’s processor is always executing the most eligible distributable thread present; the other distributable threads wait as they should. Remote invocations and returns constitute scheduling events at both source and destination nodes, and may be dealt with accordingly by the active scheduling policy.

Because Java is a multi-threaded language, it is intuitive for programmers to express applications in terms of this type of concurrency.

A distributable thread may have end-to-end time constraints. Typically, a time constraint is declared as a lexically scoped attribute of an action performed by a schedulable entity (e.g., thread). While executing within the time constraint scope, the thread might be said to be a “realtime” one, and otherwise a “non-realtime” one. When a remote invocation occurs, the platform causes a time constraint to be propagated to, and enforced on, any nodes downstream of the declaration point.

Partial Failures

Nontrivial dynamic distributed systems must be presumed always to be partially failed. At any given time, transmissions may be lost for a variety of reasons, nodes are overloaded, failing, rebooting, joining, or departing the system. Because distributable threads execute in an environment subject to partial failures typically not experienced by nodal (e.g., operating system) threads, provisions must be made to ensure the end-to-end integrity of distributable threads in a manner that assigns responsibilities appropriately between the particular system and its applications, and can be reasoned about. (The Real-Time CORBA specification leaves this issue to be addressed as added value by the ORB vendors.)

Several approaches to distributable thread integrity have been demonstrated in prior work. The DRTSJ seeks to build on this prior work 1) by providing example implementations of those existing schemes; 2) by providing APIs allowing applications to use their own integrity policies; and 3) by providing a set of example implementations from the cutting edge of research in this area, focused on thread integrity in the increasingly important field of mobile, ad-hoc networks.

Implementation Implications

Implementers are able to implement distributable threads with any transport infrastructures (RMI, SOAP, Real-Time CORBA, etc.) that provides an interface comparable to that of Java RMI's programming model. The DRTSJ distributable threads abstraction is consciously very similar to that of Real-Time CORBA 2, to facilitate application programmers' ability to write distributed programs that use both infrastructures, as has been requested by numerous prospective DRTSJ users.

2.2 Scheduling

Distributed threads carry their priority with them as they cross virtual machine boundaries. Virtual machines running on the same operating system instance should have the same mapping of Java priorities to operating system priorities. Virtual machines running on different operating system instances should have behaviorally equivalent priorities, such that threads running on two different virtual machines with the same Java priority have

similar precedence on the two machines. In both cases, it is up to the installation to insure this equivalence.

Chapter 3

The Java and Real-Time Specification for Java Platform

DRTSJ builds on top of the mechanisms of the Real-Time Specification for Java. It requires the realtime thread, asynchronous event, scheduling, and priority inversion avoidance mechanisms for proper functionality. Less emphasis is placed the memory areas of the RTSJ. To a large extent, the advent of realtime garbage collection makes these classes in their current form less relevant. Still DRTSJ should be able to work with them, even though they are not strictly required. This dependency on the RTSJ means that the two specification need to be coordinated.

Chapter 4

API Specification

4.1 Interfaces

DistributableThreadOperations

A distributable thread is a context of execution in a distributed program. Distributable threads are intended (conceptually) to generalize Thread to encompass control-flow semantics extending across multiple JVMs. This abstract definition explicitly avoids defining a particular implementation of DTs, though it is anticipated that most conforming DRTSJ implementations will employ a collection of local Thread instances as "segments" of the thread. However, the DRTSJ expert group encourages and expects implementors to provide efficient, first-class native implementations of DTs. Some terminology: We denote the object, thread segment, and JVM/node in which the DT is initially instantiated as the root of the DT. The object, thread segment, or JVM/node in which the thread is currently executing is called the head. Because partial failures are possible in distributed systems, it is challenging to enforce the kinds of invariants expected on normal Java or OS threads. Conforming implementations of DRTSJ will trade off some of the following goals.

- Instances of DistributableThread are inherently distributed objects. This means that the true state of the distributable thread may not be known, and in fact may in principle be unknowable. Implementations should aim to carefully document the relationship between the state reported by local instances of DistributableThread and the actual

state of the distributed entity.

- Enforcing a "single point of control" invariant for distributable threads is a key design goal. It may be impossible in practice to provide this exactly, but deviations from this should be documented by conforming implementations.
- Real-time operations – This abstraction has been articulated with distributed, real-time behaviors at its core. The abstraction itself has been generalized to match as closely as possible with vanilla Threads, and to admit such implementations, but the presence of real-time concerns always take precedence.
- Interaction with Java monitors
- Asynchrony
- Replaces the local thread ID with a globally-meaningful ID. Provides a few distributed operations which should be treated as explicitly unsafe.

Methods

getThreadID

```
javafx.runtime.distributed.thread.DThreadID getThreadID()
```

Get the ID of this thread.

Returns `javafx.runtime.distributed.thread.DThreadID` thread ID

trapLocal

```
void trapLocal()
```

Attempt to pause the thread. Semantics of this are currently undefined, but at the very least the local segment is paused and will make no forward progress until `unpause()` is called.

untrapLocal

```
void untrapLocal()
```

See `pause()`

run

```
void run()
```

start

```
void start()
```

getCurrentContext

```
javax.realtime.distributed.thread.DistributableThreadContext  
getCurrentContext()
```

Get the current execution context for this thread. Primarily useful for trans-node communication, and useful for distributed scheduling, TMAR, and dthread implementation.

Returns `javax.realtime.distributed.thread.DistributableThreadContext`
the current thread execution context

DistributedAsyncEventLocalMethods**bindTo**

```
void bindTo(java.lang.String happening) throws  
javax.realtime.UnknownHappeningException
```

Throws `javax.realtime.UnknownHappeningException`

DistributedAsynchronousEventHelper

Hook to manage AsyncEvent proxies to ensure they are delivered to the appropriate JVM.

Methods

createAuxilliaryAsyncEvent

```
RTRemoteAsynchronousEvent createAuxilliaryAsyncEvent()  
throws java.rmi.RemoteException
```

Throws `java.rmi.RemoteException`

DRTSJLocalServices**Methods****registerRemotelyAccessibleObject**

```
boolean  
registerRemotelyAccessibleObject(java.rmi.server.RemoteObject  
remoteName, java.lang.Object obj)
```

Register locally created Remote objects.

lookupRemotelyAccessibleObject

```
java.lang.Object  
lookupRemotelyAccessibleObject(java.rmi.server.RemoteObject  
remoteName)
```

Lookup a Remote object and return its implementation if it is local.

registerThreadEventListener

```
void  
registerThreadEventListener(javax.realtime.distributed.tmar.DistributableThr  
listener)
```

Register an event listener for distributable thread events.

registerSegmentedThreadEventListener

```
void  
registerSegmentedThreadEventListener(javax.realtime.distributed.tmar.Segment  
listener)
```

Register an event listener for distributable thread events.

DThreadSegmentOperations

Represents the local segment of a DT.

Methods

getThreadID

```
javax.realtime.distributed.thread.DThreadID getThreadID()
```

Get the ID of the enclosing thread.

Returns `javax.realtime.distributed.thread.DThreadID` thread ID

getSegmentID

```
javax.realtime.distributed.thread.DThreadSegmentID  
getSegmentID()
```

Get the unique ID for this segment.

Returns `javax.realtime.distributed.thread.DThreadSegmentID` the segment ID

getLocalSegments

```
DThreadSegment[] getLocalSegments()
```

Get all the local segments of this DT. (Not thread-safe).

Returns `DThreadSegment[]` array of segments

isRoot

```
boolean isRoot()
```

Is this currently the Root of a DT.

Returns boolean

isHead

```
boolean isHead()
```

Is this currently the Head of a DT. (Not thread safe.)

Returns boolean

getHostingNode

```
java.net.InetAddress getHostingNode()
```

Get the address of the node hosting this segment.

Returns java.net.InetAddress the address of the hosting node

getDownstreamNode

```
java.net.InetAddress getDownstreamNode()
```

Get the network address for the downstream segment. Returns null if there is no downstream segment (ie, if we are the head.)

Returns java.net.InetAddress downstream address

getUpstreamNode

```
java.net.InetAddress getUpstreamNode()
```

Get the network address for the upstream segment. Returns null if there is no upstream segment (ie, if we are the root.)

Returns java.net.InetAddress upstream address

getEnclosingThread

```
DistributableThreadOperations getEnclosingThread()
```


Returns DistributableThreadOperations**getDownstreamSegmentID**

```

    javax.realtime.distributed.thread.DThreadSegmentID
    getDownstreamSegmentID()

```

Get the segment identifier for the downstream segment. Returns null if there is no downstream segment (ie, if we are the head.)

Returns `javax.realtime.distributed.thread.DThreadSegmentID` upstream address

setDownstreamSegmentID

```

    void
    setDownstreamSegmentID(javax.realtime.distributed.thread.DThreadSegmentID
    id)

```

Set the segment identifier for the downstream segment. Returns null if there is no downstream segment (ie, if we are the head.)

getUpstreamSegmentID

```

    javax.realtime.distributed.thread.DThreadSegmentID
    getUpstreamSegmentID()

```

Get the segment identifier for the upstream segment. Returns null if there is no upstream segment (ie, if we are the root.)

Returns `javax.realtime.distributed.thread.DThreadSegmentID` upstream address

deliverPendingException

```

    boolean deliverPendingException(java.lang.Exception e)

```

Called by the runtime, or possibly by the user, in order to deliver an exception to a particular segment of the thread. This exception is guaranteed to be delivered whenever the user calls into RMI code.

pushPendingException

```
boolean pushPendingException(java.lang.Exception e)
```

Called by the runtime or by TMAR code in order to enqueue an exception in the RMI runtime for this segment. This has the effect of replacing any RMI return with the contents of the specified exception. This is dangerous, so be careful.

Parameter `java.lang.Exception e`

Returns `boolean`

forcePendingException

```
boolean forcePendingException()
```

Called by the runtime or by TMAR code in order to force an enqueued exception in the RMI runtime for this segment to be thrown immediately (if possible.) This has the effect of terminating any pending invocation and returning the throwable specified. RMI return with the contents of the specified exception. This is dangerous, so be careful.

Returns `boolean`

RealtimeRemote

`RealtimeRemote` is a marker indicating that an object provides `java.rmi.Remote`-style services while providing some level of real-time guarantees.

RTRemoteAsynchronousEvent**Methods****addHandler**

```
void addHandler(RTRemoteAsynchronousEventHandler handler)  
throws java.rmi.RemoteException
```

Parameter `RTRemoteAsynchronousEventHandler` **handler**

Throws `java.rmi.RemoteException`

fire

```
void fire() throws java.rmi.RemoteException
```

Throws `java.rmi.RemoteException`

RTRemoteAsynchronousEventHandler

Methods

getDRTSJPlatformSvcManager

```
DistributedAsynchronousEventHelper
getDRTSJPlatformSvcManager() throws
java.rmi.RemoteException
```

Throws `java.rmi.RemoteException`

4.2 Classes

DistributableThread

A distributable thread is a context of execution in a distributed program. Distributable threads are intended (conceptually) to generalize `Thread` to encompass control-flow semantics extending across multiple JVMs. This abstract definition explicitly avoids defining a particular implementation of DTs, though it is anticipated that most conforming DRTSJ implementations will employ a collection of local `Thread` instances as "segments" of the thread. However, the DRTSJ expert group encourages and expects implementors to provide efficient, first-class native implementations of DTs. Some terminology: We denote the object, thread segment, and JVM/node in which the DT is initially instantiated as the root of the DT. The object, thread segment, or JVM/node in which the thread is currently executing is called the head. Because partial failures are possible in distributed systems,

it is challenging to enforce the kinds of invariants expected on normal Java or OS threads. Conforming implementations of DRTSJ will trade off some of the following goals.

- Instances of `DistributableThread` are inherently distributed objects. This means that the true state of the distributable thread may not be known, and in fact may in principle be unknowable. Implementations should aim to carefully document the relationship between the state reported by local instances of `DistributableThread` and the actual state of the distributed entity.
- Enforcing a "single point of control" invariant for distributable threads is a key design goal. It may be impossible in practice to provide this exactly, but deviations from this should be documented by conforming implementations.
- Real-time operations – This abstraction has been articulated with distributed, real-time behaviors at its core. The abstraction itself has been generalized to match as closely as possible with vanilla `Threads`, and to admit such implementations, but the presence of real-time concerns always takes precedence.
- Interaction with Java monitors
- Asynchrony
- Replaces the local thread ID with a globally-meaningful ID.
- Provides a few distributed operations which should be treated as explicitly unsafe.

Constructors

`DistributableThread`

```
public DistributableThread(  
    javax.realtime.distributed.DistributableThreadGroup  
    group, java.lang.Runnable r, java.lang.String name)
```

Creates a new `DistributableThread` object with specified group, runnable code, and name. If group is null, the thread is assigned to the global group. [FIXME: This should be context-dependent on the security manager.] If r is not null, the specified Runnable code is used as the body of the thread when the `start()` method is called. Otherwise, the `run()` method of the thread object itself is called.

Parameter `javax.realtime.distruted group` the thread group to which the new thread will belong

Parameter `java.lang.Runnable r` the Runnable object for this thread

Parameter `java.lang.String name` the printable name of the new thread

Methods

`sleep`

```
public static void sleep(long milliseconds) throws
    java.lang.InterruptedException
```

Throws `java.lang.InterruptedException`

`sleep`

```
public static void sleep(Clock clock, HighResolutionTime
    time) throws java.lang.InterruptedException
```

Throws `java.lang.InterruptedException`

`getID`

```
public abstract GlobalIdentifier getID()
```

Get the ID of this thread.

Returns `GlobalIdentifier` thread ID

pause

```
public abstract void pause()
```

Attempt to pause the thread. Semantics of this are currently undefined, but at the very least the local segment is paused and will make no forward progress until `unpause()` is called.

unpause

```
public abstract void unpause()
```

See `pause()`

run

```
public abstract void run()
```

start

```
public abstract void start()
```

DistributedAsyncEvent**Constructors****DistributedAsyncEvent**

```
public DistributedAsyncEvent() throws  
java.rmi.RemoteException
```

Throws java.rmi.RemoteException Methods**addHandler**

```
public void addHandler(AsyncEventHandler handler)
```

bindTo

```
public void bindTo(java.lang.String happening) throws  
javax.realtime.UnknownHappeningException
```

Throws `javax.realtime.UnknownHappeningException`

fire

```
public void fire()
```

initiateFire

```
public void initiateFire() throws  
java.rmi.RemoteException
```

Throws `java.rmi.RemoteException`

DistributedAsyncEventHandler

Constructors

DistributedAsyncEventHandler

```
public DistributedAsyncEventHandler(java.lang.Runnable  
logic) throws java.rmi.RemoteException
```

Throws `java.rmi.RemoteException` **Methods**

getDRTSJPlatformSvcManager

```
public DistributedAsynchronousEventHelper  
getDRTSJPlatformSvcManager() throws  
java.rmi.RemoteException
```

Throws `java.rmi.RemoteException`

getLocalAsyncEventHandler

```
public AsyncEventHandler getLocalAsyncEventHandler()
```

DistributedRealtimeClock

Represents the single, guaranteed-available distributed clock. Expected to be equivalent to an NTP-synchronized clock, and it may be identical to the results of `javax.realtime.Clock.getRealtimeClock()`.

Constructors

DistributedRealtimeClock

```
public DistributedRealtimeClock()
```

Methods

getEpochOffset

```
public RelativeTime getEpochOffset()
```

getResolution

```
public RelativeTime getResolution()
```

getTime

```
public AbsoluteTime getTime()
```

getTime

```
public AbsoluteTime getTime(AbsoluteTime time)
```

setResolution

```
public void setResolution(RelativeTime resolution)
```

DRTSJPlatform

System-level operations for the DRTSJ platform. Many of these will be moved elsewhere over time, and protected using the Java security framework and package protections. This is a singleton object instance with a static factory method. This object instance can be used to uniquely identify a DRTSJ platform (since it is a singleton object instance and that is the

realm over which the singleton property can be enforced) and offers methods to access specific information concerning this DRTSJ platform. **FIXME:** Ensure that this is loaded by the boot classloader, or that some mechanism enforces its singleton status across all classloaders. **FIXME:** Determine which of these methods belong in `DRTSJServicesManager`.

Fields

DRTSJLevel

```
public static final int DRTSJLevel
```

DRTSJ Level of this platform

DRTSJRemoteServer

```
public static final DRTSJServicesManager
DRTSJRemoteServer
```

Methods

getInstance

```
public static final DRTSJPlatform getInstance()
```

static factory method

getDistributableThread

```
public DistributableThreadOperations
getDistributableThread(javax.realtime.distributed.thread.DThreadID
gid)
```

Return a reference to the distributable thread with the given identifier, if any is registered locally.

Parameter `javax.realtime.distributed.thread.DThreadID gid` distributable thread id

Returns `DistributableThreadOperations` reference to distributable thread

getSuccessorSegment

```
public DThreadSegmentOperations
    getSuccessorSegment(GlobalIdentifier predecessorID)
```

Return the successor segment for the given segment identifier, if it is hosted locally. Returns null if no successor is hosted locally.

Parameter GlobalIdentifier predecessorID

Returns GlobalIdentifier a reference to the local DThreadSegment

getPredecessorSegment

```
public DThreadSegmentOperations
    getPredecessorSegment(GlobalIdentifier successorID)
```

Return the predecessor segment for the given segment identifier, if it is hosted locally. Returns null if no successor is hosted locally.

Parameter GlobalIdentifier successorID

Returns DThreadSegmentOperations a reference to the local DThreadSegment

getHostedThread

```
public DistributableThreadOperations
    getHostedThread(javax.realtime.distributed.thread.DThreadID id)
```

Returns a reference to the distributable thread with the specified ID, if that thread is locally hosted.

Parameter javax.realtime.distributed.thread.DThreadID id the ID to search for

Returns DistributableThreadOperations the dthread reference, or null if it is not local

getHostedSegments

```
public DThreadSegment[] getHostedSegments()
```

Returns `DThreadSegment[]` the distributed thread segments hosted by this platform

getHostedThreads

```
public DistributableThread[] getHostedThreads()
```

Returns `DistributableThread[]` the distributed threads hosted by this platform

interruptSegment

```
public void interruptSegment(DThreadSegmentOperations  
    seg, java.lang.Exception toDeliver)
```

Called by the runtime, or possibly by the user, in order to deliver an exception to a particular segment of the thread. This exception is guaranteed to be delivered whenever the user calls into RMI code.

Parameter `DThreadSegmentOperations seg` target segment

Parameter `java.lang.Exception toDeliver` the throwable to deliver

pushSegmentPendingException

```
public void  
    pushSegmentPendingException(DThreadSegmentOperations seg,  
        javax.realtime.distributed.thread.ThreadIntegrityException  
        toDeliver)
```

Called by the runtime or by TMAR code in order to enqueue an exception in the RMI runtime for this segment. This has the effect of replacing any RMI return with the contents of the specified exception. This is dangerous, so be careful. basically there are three ways to deliver this exception:

- deliver it to a pending client invocation (it will be raised immediately)
- deliver it to a pending server invocation (it will be raised later....)
- deliver it to the segment structure itself

Perhaps this should deliver it (synchronously) to the segment itself, at which point the segment instance can delegate it to the appropriate place.

Parameter `DThreadSegmentOperations seg` the target segment

Parameter `javax.realtime.distributed.thread.ThreadIntegrityException toDeliver` the exception to deliver

`forceSegmentPendingException`

```
public void
forceSegmentPendingException(DThreadSegmentOperations
seg)
```

Called by the runtime or by TMAR code in order to force an enqueued exception in the RMI runtime for this segment to be thrown immediately (if possible.) This has the effect of terminating any pending invocation and returning the throwable specified. RMI return with the contents of the specified exception. This is dangerous, so be careful. This is only valid (and will only take effect) if `pushSegmentPendingException(DThreadSegmentOperations, ThreadIntegrityException)` has been called.

Parameter `DThreadSegmentOperations seg` the target segment

`toString`

```
public final java.lang.String toString()
```

`registerSegmentedThreadEventListener`

```
public void
registerSegmentedThreadEventListener(javax.realtime.distributed.tmar.Segment
listener)
```

Add a new segmented thread event listener. Intended for thread integrity protocols.

registerSegmentedThreadIDEventListener

```
public void
registerSegmentedThreadIDEventListener(javax.realtime.distributed.tmar.SegmentedThreadIDEventListener listener)
```

Add a new segmented thread event listener. Intended for thread integrity protocols.

eventDTInvokeArrived

```
public void
eventDTInvokeArrived(DistributableThreadOperations thread, DThreadSegmentOperations segment)
```

Notify all registered listeners that the specified thread has arrived. See `SegmentedDTEventListener`.

Parameter `DistributableThreadOperations thread` the arriving thread

Parameter `DThreadSegmentOperations segment` the arriving thread's local segment

eventDTInvokeDeparted

```
public void
eventDTInvokeDeparted(DistributableThreadOperations thread, DThreadSegmentOperations segment)
```

Notify all registered listeners that the specified thread has departed. See `SegmentedDTEventListener`.

Parameter `DistributableThreadOperations thread` the arriving thread

Parameter `DThreadSegmentOperations segment` the arriving thread's local segment

eventDTReturnArrived

```
public void
eventDTReturnArrived(DistributableThreadOperations
thread, DThreadSegmentOperations segment)
```

Notify all registered listeners that the specified thread has arrived in return context. See `SegmentedDTEEventListener`.

Parameter `DistributableThreadOperations thread` the arriving thread

Parameter `DThreadSegmentOperations segment` the arriving thread's local segment

eventDTReturnDeparted

```
public void
eventDTReturnDeparted(DistributableThreadOperations
thread, DThreadSegmentOperations segment)
```

Notify all registered listeners that the specified thread has departed in return context. See `SegmentedDTEEventListener`.

Parameter `DistributableThreadOperations thread` the departing thread

Parameter `DThreadSegmentOperations segment` the departing thread's local segment

eventNewRootSegment

```
public void
eventNewRootSegment(DistributableThreadOperations thread,
DThreadSegmentOperations segment)
```

Notify all registered listeners that a new root thread segment has been created. See `SegmentedDTEEventListener`.

Parameter `DistributableThreadOperations thread` the departing thread

Parameter DThreadSegmentOperations segment the departing thread's local segment

getIntegrityManager

```
public
  javax.realtime.distributed.tmar.ThreadIntegrityManager
  getIntegrityManager()
```

setIntegrityManager

```
public void
  setIntegrityManager(javax.realtime.distributed.tmar.ThreadIntegrityManager
  integrityManager)
```

trapDistributableThread

```
public DistributableThreadOperations
  trapDistributableThread(javax.realtime.distributed.thread.DThreadID
  toPause)
```

Note: this is a very unsafe operation with poorly defined semantics.

Returns DistributableThreadOperations

untrapDistributableThread

```
public void
  untrapDistributableThread(javax.realtime.distributed.thread.DThreadID
  toPause)
```

Note: this is a very unsafe operation with poorly defined semantics.

DRTSJServicesManager

This object instance offers (remote) access to specific DRTSJ-related methods. This is a singleton object instance with a static factory method.

Methods

getInstance

```
public static DRTSJServicesManager getInstance()
```

createAuxilliaryAsyncEvent

```
public RTRemoteAsynchronousEvent  
createAuxilliaryAsyncEvent() throws  
java.rmi.RemoteException
```

Throws `java.rmi.RemoteException`

registerRemotelyAccessibleObject

```
public boolean  
registerRemotelyAccessibleObject(java.rmi.server.RemoteObject  
remoteName, java.lang.Object obj)
```

Description copied from interface: DRTSJLocalServices Register locally created Remote objects.

lookupRemotelyAccessibleObject

```
public java.lang.Object  
lookupRemotelyAccessibleObject(java.rmi.server.RemoteObject  
remoteName)
```

Description copied from interface: DRTSJLocalServices Lookup a Remote object and return its implementation if it is local.

registerSegmentedThreadEventListener

```
public void  
registerSegmentedThreadEventListener(javax.realtime.distributed.tmar.Segment  
listener)
```

Add a new segmented thread event listener. Intended for thread integrity protocols.

registerSegmentedThreadIDEventListener

```
public void
registerSegmentedThreadIDEventListener(javax.realtime.distributed.tmar.SegmentedThreadIDEventListener listener)
```

Add a new segmented thread event listener. Intended for thread integrity protocols.

registerThreadEventListener

```
public void
registerThreadEventListener(javax.realtime.distributed.tmar.DistributableThreadEventListener listener)
```

Description copied from interface: `DRTSJLocalServices` Register an event listener for distributable thread events.

DThreadSegment

Represents the local segment of a DT.

Constructors**DThreadSegment**

```
public DThreadSegment()
```

Methods**getThreadID**

```
public abstract
javax.realtime.distributed.thread.DThreadID getThreadID()
```

Get the ID of the enclosing thread.

Returns `javax.realtime.distributed.thread.DThreadID` thread ID

getSegmentID

```
public abstract
    javax.realtime.distributed.thread.DThreadSegmentID
    getSegmentID()
```

Get the unique ID for this segment.

Returns `javax.realtime.distributed.thread.DThreadSegmentID` the segment ID

GlobalIdentifier

An identifier which promises to be globally unique on the network.

Constructors**GlobalIdentifier**

```
public GlobalIdentifier()
```

GlobalIdentifier

```
public GlobalIdentifier(java.rmi.server.UID inputID,
    java.lang.String host)
```

Methods**hashCode**

```
public int hashCode()
```

getHostAddress

```
public byte[] getHostAddress()
```

getLocalID

```
public java.rmi.server.UID getLocalID()
```

read

```
public void read(java.io.ObjectInput oin) throws
    java.io.IOException
```

Throws java.io.IOException

write

```
public void write(java.io.ObjectOutput oout) throws
    java.io.IOException
```

Throws java.io.IOException

equals

```
public boolean equals(java.lang.Object obj)
```

fromHex

```
public java.lang.String fromHex(byte[] source)
```

toString

```
public java.lang.String toString()
```

printID

```
public java.lang.String printID()
```

readExternal

```
public void readExternal(java.io.ObjectInput input)
    throws java.io.IOException,
    java.lang.ClassNotFoundException
```

Throws java.io.IOException

Throws java.lang.ClassNotFoundException

writeExternal

```
public void writeExternal(java.io.ObjectOutput output)
    throws java.io.IOException
```

Throws `java.io.IOException`

GlobalRealtimeThread

A Level-1 approximation of a distributable thread.

Constructors**GlobalRealtimeThread**

```
public
GlobalRealtimeThread(javax.realtime.distributed.thread.RealtimeThreadContext
    sourceContext, java.lang.Runnable r)
```

Magical constructor for use when instantiating a new segment of a `DistributableThread`.

Parameter `javax.realtime.distributed.thread.RealtimeThreadContext`
`sourceContext`

Parameter `java.lang.Runnable` `r`

GlobalRealtimeThread

```
public GlobalRealtimeThread()
```

GlobalRealtimeThread

```
public
GlobalRealtimeThread(javax.realtime.distributed.thread.DThreadID
    ID)
```

GlobalRealtimeThread

```
public GlobalRealtimeThread(SchedulingParameters
    scheduling, ReleaseParameters release, MemoryParameters
    memory, MemoryArea area, ProcessingGroupParameters group,
    java.lang.Runnable r)
```

GlobalRealtimeThread

```
public GlobalRealtimeThread(SchedulingParameters
    scheduling, ReleaseParameters release, MemoryParameters
    memory, MemoryArea area, ProcessingGroupParameters group,
    java.lang.Runnable r,
    javax.realtime.distributed.thread.DThreadID ID)
```

Methods**setGlobalThreadIdentity**

```
public void
    setGlobalThreadIdentity(javax.realtime.distributed.thread.RealtimeThreadCont
    c)
```

setInvocationTimeout

```
public void setInvocationTimeout(RelativeTime timeout)
```

setAckTimeout

```
public void setAckTimeout(RelativeTime timeout)
```

setInvocationTimeout

```
public RelativeTime setInvocationTimeout()
```

setAckTimeout

```
public RelativeTime setAckTimeout()
```

start

```
public void start()
```

hashCode

```
public int hashCode()
```

equals

```
public boolean equals(java.lang.Object obj)
```

getThreadID

```
public javax.realtime.distributed.thread.DThreadID  
getThreadID()
```

Description copied from interface: `DistributableThreadOperations` Get the ID of this thread.

Returns `javax.realtime.distributed.thread.DThreadID` thread ID

trapLocal

```
public void trapLocal()
```

Description copied from interface: `DistributableThreadOperations` Attempt to pause the thread. Semantics of this are currently undefined, but at the very least the local segment is paused and will make no forward progress until `unpause()` is called.

untrapLocal

```
public void untrapLocal()
```

Description copied from interface: `DistributableThreadOperations` See `pause()`

setReleaseParametersIfFeasible

```
public boolean
setReleaseParametersIfFeasible(ReleaseParameters
releaseparameters)
```

getCurrentContext

```
public
javax.realtime.distributed.thread.DistributableThreadContext
getCurrentContext()
```

Description copied from interface: `DistributableThreadOperations` Get the current execution context for this thread. Primarily useful for trans-node communication, and useful for distributed scheduling, TMAR, and `dthread` implementation.

Returns `javax.realtime.distributed.thread.DistributableThreadContext` the current thread execution context

getSegmentID

```
public javax.realtime.distributed.thread.DThreadSegmentID
getSegmentID()
```

Description copied from interface: `DThreadSegmentOperations` Get the unique ID for this segment.

Returns `javax.realtime.distributed.thread.DThreadSegmentID` the segment ID

getLocalSegments

```
public DThreadSegment[] getLocalSegments()
```

Description copied from interface: `DThreadSegmentOperations` Get all the local segments of this DT. (Not thread-safe).

Returns `DThreadSegment[]` array of segments

isRoot

```
public boolean isRoot()
```

Description copied from interface: `DThreadSegmentOperations` Is this currently the Root of a DT.

Returns boolean

isHead

```
public boolean isHead()
```

Description copied from interface: `DThreadSegmentOperations` Is this currently the Head of a DT. (Not thread safe.)

Returns boolean

getHostingNode

```
public java.net.InetAddress getHostingNode()
```

Description copied from interface: `DThreadSegmentOperations` Get the address of the node hosting this segment.

Returns java.net.InetAddress the address of the hosting node

getDownstreamNode

```
public java.net.InetAddress getDownstreamNode()
```

Description copied from interface: `DThreadSegmentOperations` Get the network address for the downstream segment. Returns null if there is no downstream segment (ie, if we are the head.)

Returns java.net.InetAddress downstream address

getUpstreamNode

```
public java.net.InetAddress getUpstreamNode()
```

Description copied from interface: `DThreadSegmentOperations` Get the network address for the upstream segment. Returns null if there is no upstream segment (ie, if we are the root.)

Returns `java.net.InetAddress` upstream address

getEnclosingThread

```
public DistributableThreadOperations getEnclosingThread()
```

Returns `DistributableThreadOperations`

getDownstreamSegmentID

```
public javax.realtime.distributed.thread.DThreadSegmentID
getDownstreamSegmentID()
```

Description copied from interface: `DThreadSegmentOperations` Get the segment identifier for the downstream segment. Returns null if there is no downstream segment (ie, if we are the head.)

Returns `javax.realtime.distributed.thread.DThreadSegmentID` upstream address

getUpstreamSegmentID

```
public javax.realtime.distributed.thread.DThreadSegmentID
getUpstreamSegmentID()
```

Description copied from interface: `DThreadSegmentOperations` Get the segment identifier for the upstream segment. Returns null if there is no upstream segment (ie, if we are the root.)

Returns `javax.realtime.distributed.thread.DThreadSegmentID` upstream address

readExternal

```
public void readExternal(java.io.ObjectInput input)
    throws java.io.IOException,
           java.lang.ClassNotFoundException
```

Throws java.io.IOException

Throws java.lang.ClassNotFoundException

writeExternal

```
public void writeExternal(java.io.ObjectOutput output)
    throws java.io.IOException
```

Throws java.io.IOException

deliverPendingException

```
public boolean
    deliverPendingException(java.lang.Exception e)
```

Description copied from interface: DThreadSegmentOperations Called by the runtime, or possibly by the user, in order to deliver an exception to a particular segment of the thread. This exception is guaranteed to be delivered whenever the user calls into RMI code.

pushPendingException

```
public boolean pushPendingException(java.lang.Exception
    e)
```

Description copied from interface: DThreadSegmentOperations Called by the runtime or by TMAR code in order to enqueue an exception in the RMI runtime for this segment. This has the effect of replacing any RMI return with the contents of the specified exception. This is dangerous, so be careful.

Returns boolean

forcePendingException

```
public boolean forcePendingException()
```

Description copied from interface: `DThreadSegmentOperations` Called by the runtime or by TMAR code in order to force an enqueued exception in the RMI runtime for this segment to be thrown immediately (if possible.) This has the effect of terminating any pending invocation and returning the throwable specified. RMI return with the contents of the specified exception. This is dangerous, so be careful.

Returns boolean**isExternalTMAR**

```
public boolean isExternalTMAR()
```

setExternalTMAR

```
public void setExternalTMAR(boolean externalTMAR)
```

setDownstreamSegmentID

```
public void  
setDownstreamSegmentID(javax.realtime.distributed.thread.DThreadSegmentID  
id)
```

Description copied from interface: `DThreadSegmentOperations` Set the segment identifier for the downstream segment. Returns null if there is no downstream segment (ie, if we are the head.)

hasPendingException

```
public boolean hasPendingException()
```

getPendingException

```
public java.lang.Exception getPendingException()
```

setTMARPolicy

```
public void
setTMARPolicy(javax.realtime.distributed.tmar.TMARPolicyParameters
policy)
```

getTMARPolicy

```
public
javax.realtime.distributed.tmar.TMARPolicyParameters
getTMARPolicy()
```

GlobalThread

A Level-1 approximation of a distributable thread.

Constructors**GlobalThread**

```
public
GlobalThread(javax.realtime.distributed.thread.DistributableThreadContext
sourceContext, java.lang.Runnable r)
```

Magical constructor for use when instantiating a new segment of a `DistributableThread`.

Parameter `javax.realtime.distributed.thread.DistributableThreadContext`
`sourceContext`

Parameter `java.lang.Runnable` `r`

GlobalThread

```
public GlobalThread()
```

GlobalThread

```
public GlobalThread(java.lang.Runnable r)
```

Methods

getGlobalID

```
public javax.realtime.distributed.thread.DThreadID
getGlobalID()
```

Return this thread's ID.

Returns `javax.realtime.distributed.thread.DThreadID` global thread identifier

setGlobalID

```
public void
setGlobalID(javax.realtime.distributed.thread.DThreadID
id)
```

Do we really need to be able to set the GlobalID of a thread? added against all odds by JA. maybe unnecessary.

setGlobalThreadIdentity

```
public void
setGlobalThreadIdentity(javax.realtime.distributed.thread.RealtimeThreadCont
c)
```

relinquishIdentity

```
public void relinquishIdentity()
```

writeEssence

```
public void writeEssence(java.io.ObjectOutputStream oout)
throws java.io.IOException
```

Throws `java.io.IOException`

readEssence

```
public void readEssence(java.io.ObjectInputStream oin)
throws java.io.IOException
```

Throws `java.io.IOException`

hashCode

```
public int hashCode()
```

equals

```
public boolean equals(java.lang.Object obj)
```

pause

```
public void pause()
```

unpause

```
public void unpause()
```

getCurrentContext

```
public  
javax.realtime.distributed.thread.DistributableThreadContext  
getCurrentContext()
```

Description copied from interface: `DistributableThreadOperations` Get the current execution context for this thread. Primarily useful for trans-node communication, and useful for distributed scheduling, TMAR, and `dthread` implementation.

Returns `javax.realtime.distributed.thread.DistributableThreadContext`
the current thread execution context

getThreadID

```
public javax.realtime.distributed.thread.DThreadID  
getThreadID()
```

Description copied from interface: `DistributableThreadOperations` Get the ID of this thread.

Returns `javax.realtime.distributed.thread.DThreadID` thread ID

getSegmentID

```
public javax.realtime.distributed.thread.DThreadSegmentID
    getSegmentID()
```

Description copied from interface: `DThreadSegmentOperations` Get the unique ID for this segment.

Returns `javax.realtime.distributed.thread.DThreadSegmentID` the segment ID

getLocalSegments

```
public DThreadSegment[] getLocalSegments()
```

Description copied from interface: `DThreadSegmentOperations` Get all the local segments of this DT. (Not thread-safe).

Returns `DThreadSegment[]` array of segments

isRoot

```
public boolean isRoot()
```

Description copied from interface: `DThreadSegmentOperations` Is this currently the Root of a DT.

Returns `boolean`

isHead

```
public boolean isHead()
```

Description copied from interface: `DThreadSegmentOperations` Is this currently the Head of a DT. (Not thread safe.)

Returns `boolean`

getHostingNode

```
public java.net.InetAddress getHostingNode()
```

Description copied from interface: `DThreadSegmentOperations` Get the address of the node hosting this segment.

Returns `java.net.InetAddress` the address of the hosting node

getDownstreamNode

```
public java.net.InetAddress getDownstreamNode()
```

Description copied from interface: `DThreadSegmentOperations` Get the network address for the downstream segment. Returns null if there is no downstream segment (ie, if we are the head.)

Returns `java.net.InetAddress` downstream address

getUpstreamNode

```
public java.net.InetAddress getUpstreamNode()
```

Description copied from interface: `DThreadSegmentOperations` Get the network address for the upstream segment. Returns null if there is no upstream segment (ie, if we are the root.)

Returns `java.net.InetAddress` upstream address

getEnclosingThread

```
public DistributableThreadOperations getEnclosingThread()
```

Returns `DistributableThreadOperations`

getDownstreamSegmentID

```
public javax.realtime.distributed.thread.DThreadSegmentID  
getDownstreamSegmentID()
```

Description copied from interface: `DThreadSegmentOperations` Get the segment identifier for the downstream segment. Returns null if there is no downstream segment (ie, if we are the head.)

Returns `javax.realtime.distributed.thread.DThreadSegmentID` upstream address

getUpstreamSegmentID

```
public javax.realtime.distributed.thread.DThreadSegmentID  
getUpstreamSegmentID()
```

Description copied from interface: `DThreadSegmentOperations` Get the segment identifier for the upstream segment. Returns null if there is no upstream segment (ie, if we are the root.)

Returns `javax.realtime.distributed.thread.DThreadSegmentID` upstream address

readExternal

```
public void readExternal(java.io.ObjectInput input)  
throws java.io.IOException,  
java.lang.ClassNotFoundException
```

Throws `java.io.IOException`

Throws `java.lang.ClassNotFoundException`

writeExternal

```
public void writeExternal(java.io.ObjectOutput output)  
throws java.io.IOException
```

Throws java.io.IOException**deliverPendingException**

```
public boolean  
deliverPendingException(java.lang.Exception e)
```

Description copied from interface: DThreadSegmentOperations Called by the runtime, or possibly by the user, in order to deliver an exception to a particular segment of the thread. This exception is guaranteed to be delivered whenever the user calls into RMI code.

forcePendingException

```
public boolean forcePendingException()
```

Description copied from interface: DThreadSegmentOperations Called by the runtime or by TMAR code in order to force an enqueued exception in the RMI runtime for this segment to be thrown immediately (if possible.) This has the effect of terminating any pending invocation and returning the throwable specified. RMI return with the contents of the specified exception. This is dangerous, so be careful.

Returns boolean**pushPendingException**

```
public boolean pushPendingException(java.lang.Exception  
e)
```

Description copied from interface: DThreadSegmentOperations Called by the runtime or by TMAR code in order to enqueue an exception in the RMI runtime for this segment. This has the effect of replacing any RMI return with the contents of the specified exception. This is dangerous, so be careful.

Returns boolean

setDownstreamSegmentID

```
public void set
```

```
    DownstreamSegmentID(javax.realtime.distributed.thread.DThreadSegmentID  
    id)
```

Description copied from interface: `DThreadSegmentOperations` Set the segment identifier for the downstream segment. Returns null if there is no downstream segment (ie, if we are the head.)

trapLocal

```
public void trapLocal()
```

Description copied from interface: `DistributableThreadOperations` Attempt to pause the thread. Semantics of this are currently undefined, but at the very least the local segment is paused and will make no forward progress until `unpause()` is called.

untrapLocal

```
public void untrapLocal()
```

Description copied from interface: `DistributableThreadOperations` See `pause()`

RealtimeRemoteObject**Fields****serialVersionUID**

```
public static final long serialVersionUID
```

See Also: Constant Field Values

ref

```
protected transient java.rmi.server.RemoteRef ref
```

Constructors**RealtimeRemoteObject**

```
protected RealtimeRemoteObject()
```

RealtimeRemoteObject

```
protected RealtimeRemoteObject(java.rmi.server.RemoteRef  
newref)
```

Methods**getRef**

```
public java.rmi.server.RemoteRef getRef()
```

toStub

```
public static java.rmi.Remote toStub(java.rmi.Remote obj)  
throws java.rmi.NoSuchObjectException
```

This comes from the RemoteObject specification. In our case, we delegate the conversion to the RMIRuntime. The RMIRuntime should hold a table mapping local Remote servers to RemoteRefs. This will need to be revisited if we want to intelligently handle multiple exports. At this point, we will not support multiple exports.

Parameter java.rmi.Remote obj a Remote value

Returns java.rmi.Remote a Remote value

Throws java.rmi.NoSuchObjectException if an error occurs

hashCode

```
public int hashCode()
```

equals

```
public boolean equals(java.lang.Object obj)
```

Two RemoteObjects are considered equal if a) they are the same object, or b) they refer to the same remote object. In the latter case, we compare references using their remoteEquals method.

Parameter java.lang.Object obj an Object value

Returns boolean true when the two objects are the same.

toString

```
public java.lang.String toString()
```

getDGC

```
public java.rmi.dgc.DGC getDGC()
```

Returns a DGCImpl_Stub object that points at the DGC for this object

sendDGCack

```
public void sendDGCack(java.rmi.server.UID callID)
```

4.3 Exceptions

ThreadIntegrityViolation

A exceptional condition has occurred in a DistributableThread which requires application attention. This is an unchecked exception.

Constructors

ThreadIntegrityViolation

```
public ThreadIntegrityViolation()
```

Chapter 5

Reference Implementation

The EDR#1 software suite consists of a modified RTSJ-compliant J2ME virtual machine and a class library consisting of modified RTSJ classes as well as new classes in the `javax.realtime.distributed` package. In addition, a tested realtime Linux configuration and demonstration application are provided.

Realtime RMI

The DRTSJ RI provides a full Java Remote Method Invocation (RMI) stack, called RT-RMI, intended for use in RTSJ virtual machines. RT-RMI is wire-protocol compatible with Sun's JDK 1.4 RMI implementation, while providing extended wire protocols for invocations between DRTSJ-compliant JVMs. A datagram-based RMI wire protocol with application-level reliability mechanisms has been provided in order to demonstrate and test DRTSJ applications in dynamic and mobile, ad-hoc networks where TCP is a poor engineering solution.

RT-RMI has organic support for carrying arbitrary invocation contexts across nodes, facilitating the construction of distributable threads and potentially other end-to-end programming abstractions. By default, invocations between RT-RMI-capable nodes carry their execution context with them. Behaviors analogous to Level 2 Integration as discussed in [WCJW02] are provided transparently across the system.

The Sun Microsystems RMI wire protocol specification [Sun02c] relies heavily on Java Object Serialization [Sun02b]. However, other implementations such as RMI-IIOP [Sun02d] provide their own object marshalling

facilities. The EDR#1 specification requires the Java RMI programming model, but has been decoupled from particular Java RMI implementations. realtime object serialization and the accompanying memory allocation behaviors are left as a quality of implementation issue.

RT-RMI as provided in EDR#1 defines several new exceptions subclassed from `RemoteException` and `RuntimeException` to indicate failure conditions resulting from violations of end-to-end time constraints or thread integrity events. These exceptions should be caught and dealt with by application code; however the safety and consistency of the distributed system is preserved even if the application fails to deal with these events.

Distributable Threads

The DRTSJ implementation team has implemented distributable threads capable of interoperating with various JVMs and transport infrastructures. These threads are capable of traversing nodes with standard, RTSJ, and DRTSJ virtual machines, yielding the best available timeliness behavior feasible on each participant. Invocations and returns are presented to the programmer in a manner congruent with the RMI programming model, but have been decoupled from particular RMI implementations to the extent possible.

Thread Integrity

The DRTSJ RI provides: example implementations of prior art integrity (e.g., orphan detection and elimination) policies; APIs allowing applications to provide their own integrity policies; example implementations of new research focused on thread integrity in mobile, ad-hoc networks. We refer to the class of thread integrity protocols implemented to date as thread maintenance and repair (TMAR) protocols.

The following protocols have been implemented and will appear in the preliminary DRTSJ RI:

- *Thread Polling*, a protocol originally implemented in the Alpha research OS kernel
- A *fast failure detector* (FFD) driven TMAR, which detects link failures immediately and triggers orphan cleanup in the event of down/upstream

failures. This policy provides best-effort ordered orphan cleanup¹ if requested

- *TPR*, an approach which provides deterministic detection and cleanup times for failed distributable threads with failure handlers [CARJ06]
- *D-TPR*, an evolving algorithm and protocol for predictable detection and cleanup times in wireless and dynamic networks [Cur06]
- *W-TPR*, an evolving algorithm and protocol for predictable detection and cleanup times in wireless and dynamic networks [Cur06]

In addition, the RI may implement *Node-Alive* [GGC⁺95], a more conservative approach targeted for local area networks and very high reliability.

Pluggable Scheduling

Implementations of example distributed realtime applications and high-quality thread integrity mechanisms require support from scheduling policies. To facilitate experimentation and the construction of an acceptable RI, the implementation team has included an optional *Metascheduler* component, allowing arbitrary user-defined scheduling disciplines to be defined. While the RTSJ does specify interfaces which schedulers and schedulable objects must meet, it does not provide the primitives necessary to implement scheduling policies without the cooperation of the RTJVM vendor [DW04, ZW06].

The *Metascheduler* implements an abstract scheduling framework intended to support pluggable schedulers consistent with the RTSJ vision. While EDR#1 does not yet propose an API, the framework and *Metascheduler* are included in the RI.

A variety of scheduling disciplines have been implemented, ranging from simple, traditional (e.g., fixed priority, EDF), to *Time-Utility Function/Utility-Accrual* (TUF/UA) policies. In particular, the RI demonstrates a combined TUF/UA scheduling and thread integrity mechanism for providing bounded-time, end-to-end thread failure detection and recovery. [CARJ06]

The scheduling framework and *Metascheduler* is inspired primarily by prior work in scheduling frameworks in the Alpha research OS kernel [CJR92],

¹There is some confusion regarding the definition of *best-effort*. This specification use it in the conventional sense: the TMAR protocol attempts to provide ordered cleanup within a reasonable time constraint, but no guarantees are made to the application.

the Open Group Research Institute Mk7.3a OS integrated Alpha/Mach kernel [Ope98], and in particular the local [LR⁺04] and distributed threads [LRCJ04] Metascheduler work at Virginia Tech.

The DRTSJ RI Distribution

The DRTSJ RI and TCK will be delivered in two forms: first, traditional tarball and JAR files appropriate for cross-platform evaluation and use by JCP members; second, because of the complexity and inherent dependencies, a set of Debian packages is being maintained to streamline the “getting started” process. A package repository containing the DRTSJ core libraries, with references to all required dependencies will be provided, and constructing a test system will be simplified to a single Debian “`apt-get`” command.

The DRTSJ EDR#1 RI consists of

- a set of class libraries implementing the DRTSJ APIs and
- a set of external dependencies, including the Apache build environment and a Debian Linux system with Linux Kernel 2.6, patched with the most recent realtime extensions.

Demonstration Application

Virginia Tech has written a demonstration application [AR06] to help prospective users better understand the DRTSJ. The demonstration application is also providing essential feedback to the team designing and implementing the DRTSJ and RI. That work was performed in an ONR-funded Advanced Wireless Integrated Navy Network (AWINN) project at Virginia Tech. [AWI] Both the AWINN project and MITRE’s DRTSJ focuses are on mobile, ad hoc wireless networks (MANETs) with end-to-end time constraints.

The demonstration consists of a coastal air defense simulation, a non-trivial application written on the EDR#1 Reference Implementation, using distributable threads as the end-to-end programming and scheduling abstraction. The application consists of a collection of distributed components for managing on-board sensors, fighter/interceptors, tracking systems, and command and control C2 operations in a multi-ship naval warfare simulation. The simulation testbed includes thirteen nodes comprising a scenario

generator, a MANET/dynamic network simulator, and four communications/routing nodes, and seven application nodes running DRTSJ application code atop Linux 2.6 with realtime extensions. Novel approaches to enforcing distributable thread integrity are demonstrated and evaluated against mission metrics.

The demonstration currently exercises TMAR protocols and accompanying scheduling algorithms which provide probabilistic timing assurances for end-to-end thread behavior in the presence of application- and MANET-induced run-time uncertainties. These uncertainties include those induced by workloads, node/link failures, message losses, and node membership changes (previously open problems).

This demonstration application will be provided with the EDR#1 RI in order to aid first-time users and illustrate how a non-trivial system may be constructed using the DRTSJ.

Bibliography

- [AR06] Jonathan Anderson and Binoy Ravindran. AWINN task 2.2 final demonstration: A coastal air defense scenario. [Presentation to USN Office of Naval Research, August 2006], August 2006.
- [AWI] Advanced Wireless Integrated Navy Network (AWINN) homepage. <http://awinn.ece.vt.edu>.
- [BBD⁺06] Greg Bollella, Ben Brosgol, Peter Dibble, Steve Furr, James Gosling, David Hardin, Mark Turnbull, Rudy Belliardi, David Holmes, and Andy Wellings. The real-time specification for Java (version 1.0.2). Specification JSR-1, Java Community Process, 2006. Available: http://www.rtsj.org/specjavadoc/book_index.html.
- [CARJ06] Edward Curley, Jonathan Anderson, Binoy Ravindran, and E. Douglas Jensen. Recovering from distributable thread failures with assured timeliness in real-time distributed systems. In *Proceedings of the 2006 SRDS*, October 2006. [To Appear] Available: <http://www.real-time.ece.vt.edu/srds06.pdf>.
- [CJR92] R. K. Clark, E. D. Jensen, and F. D. Reynolds. An architectural overview of the Alpha real-time distributed kernel. In *Proceedings of the USENIX Workshop on Microkernels and Other Kernel Architectures*, April 1992.
- [Cur06] Edward Curley. Integrity assurances for distributable real-time threads in dynamic networks. Master's thesis, Virginia Polytechnic and State University, September 2006. [Anticipated].

- [Dis] DRTSJ public web site. <http://drtsj.org>.
- [DW04] Peter Dibble and Andy Wellings. The real-time specification for Java: Current status and future work. In *Proceedings of the Seventh IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 71–77, May 2004.
- [GGC⁺95] J. Goldberg, I. Greenberg, R. K. Clark, E. D. Jensen, K. Kim, and D. M. Wells. Adaptive fault-resistant systems (chapter 5: Adaptive distributed thread integrity). Technical Report csl-95-02, Computer Science Laboratory, SRI International, Menlo Park, CA., January 1995. <http://www.csl.sri.com/papers/sri-csl-95-02/>.
- [Jen02] E. Douglas Jensen. Rationale for the direction of the distributed real-time specification for Java panel position paper. In *Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2002*, pages 258–259, 2002.
- [JJ] JSR-50 Expert Group and E. Douglas Jensen. JSR-50 proposal. <http://jcp.org/en/jsr/detail?id=1>.
- [JN90] E. D. Jensen and J. D. Northcutt. Alpha: A non-proprietary operating system for large, complex, distributed real-time systems. In *IEEE Workshop on Experimental Distributed Systems*, pages 35–41, 1990.
- [JSRa] JSR-1 Expert Group. JSR-1 proposal: Real-time specification for Java. <http://jcp.org/en/jsr/detail?id=1>.
- [JSRb] JSR-282 Expert Group. JSR-282 proposal: Real-time specification for Java version 1.1. <http://jcp.org/en/jsr/detail?id=1>.
- [Lam83] Butler W. Lampson. Hints for computer system design. In *SOSP '83: Proceedings of the ninth ACM symposium on Operating systems principles*, pages 33–48, New York, NY, USA, 1983. ACM Press. Available: <http://research.microsoft.com/~lampson/33-Hints/WebPage.html>.

- [LR⁺04] P. Li, B. Ravindran, et al. A formally verified application-level framework for real-time scheduling on POSIX real-time operating systems. *IEEE Trans. Software Engineering*, 30(9):613 – 629, Sept. 2004.
- [LRCJ04] P. Li, B. Ravindran, H. Cho, and E. D. Jensen. Scheduling distributable real-time threads in Tempus middleware. In *IEEE Conference on Parallel and Distributed Systems*, pages 187 – 194, July 2004.
- [Nor87] J. D. Northcutt. *Mechanisms for Reliable Distributed Real-Time Operating Systems — The Alpha Kernel*. Academic Press, 1987.
- [Obj01] Object Management Group. Dynamic scheduling real-time CORBA 2.0 (joint revised submission), 2001. orbos/2001-04-01 ed.
- [Obj04] Object Management Group. Extensible transport framework specification – final adopted specification, 2004. ptc/04-03-03 ed.
- [Obj05] Object Management Group. Data distribution service for real-time systems, v1.1, 2005. formal/2005-12-04.
- [OMG01] OMG. Real-time CORBA 2.0: Dynamic scheduling specification. Technical report, Object Management Group, September 2001. OMG Final Adopted Specification, <http://www.omg.org/docs/ptc/01-08-34.pdf>.
- [Ope98] Open Group Research Institute’s Real-Time Group. *MK7.3a Release Notes*. The Open Group Research Institute, Cambridge, Massachusetts, October 1998. Available: <http://www.real-time.org/docs/RelNotes7.Book.pdf>.
- [Rea] Virginia Tech real-time laboratory publications site. <http://www.real-time.ece.vt.edu/papers.html>.
- [rts] RTSJ public web site. <http://rtsj.org>.

- [SG01] Umar Saif and David J. Greaves. Communication primitives for ubiquitous systems or RPC considered harmful. In *21st International Conference on Distributed Computing Systems Workshops (ICDCSW '01)*, 2001.
- [SRC84] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2(4):277–288, 1984.
- [Sun02a] Sun Microsystems. Java™ message service specification final release 1.1, April 2002. Available: <http://java.sun.com/products/jms/>.
- [Sun02b] Sun Microsystems. Java™ object serialization specification. Technical report, Sun Microsystems, 4150 Network Circle, Santa Clara, CA, November 2002. [Revision 1.4.4] Available: <http://java.sun.com/j2se/1.4/pdf/serial-spec.pdf>.
- [Sun02c] Sun Microsystems. Java™ remote method invocation specification. Technical report, Sun Microsystems, 4150 Network Circle, Santa Clara, CA, November 2002. [Revision 1.9, Java™ 2 SDK SE, v.1.4.2] Available: <http://java.sun.com/j2se/1.4/pdf/rmi-spec-1.4.2.pdf>.
- [Sun02d] Sun Microsystems. Java™ RMI over IIOP technology documentation home page. Technical report, Sun Microsystems, 4150 Network Circle, Santa Clara, CA, November 2002. [From J2SDK 1.4.2 Release Notes] Available: <http://java.sun.com/j2se/1.4.2/docs/guide/rmi-iiop/index.html>.
- [WCJW02] Andy Wellings, Raymond K. Clark, E. Douglas Jensen, and Doug Wells. A framework for integrating the Real-Time Specification for Java and Java's remote method invocation. In *Proc. of the 5th IEEE International Symposium on Object Oriented Real-Time Distributed Computing*, April 2002. Available: http://www.real-time.org/docs/isorc02_v41.pdf.
- [WWWK94] Jim Waldo, Geoff Wyant, Ann Wollrath, and Sam Kendall. A note on distributed computing. Note SMLI TR-94-29, Sun Microsystems Laboratories, Inc., 2550 Garcia Avenue, Mountain View, VA 94043, November 1994.

- [ZW06] A. Zerzelidis and A. J. Wellings. Getting more flexible scheduling in the rtsj. In *Proceedings 9th IEEE ISORC*, pages 3–10. IEEE Computer Society TC on Distributed Processing, IEEE Computer Society, April 2006.