HLA provider

HLA gateway

Version 1.0.0

User’s guide

July 17, 2012

[Remarks]
The development of this provider is supported by Japan Society for the Promotion of Machine Industry, JSPMI.

[Revision history]

Version	Date	Contents
1. 0. 0. 0	2006-02-24	First edition.
1. 0. 0. 1	2010-02-11	Added error codes.
1. 0. 0. 2	2011-03-11	Added provider registration tool related information.
1. 0. 0	2012-07-17	Changed the version management rule of the document.

[Supported model]

Model	Version	Remarks
Pitch AB	1. 3	Operation check was done with pRTI 1.3 of Pitch AB (http://www. pitch. se/).

Contents

1. Introduction

This is a user's guide for a gateway provider (hereafter, HLA provider) that establishes a connection between ORiN2 CAO (Controller Access Object) and HLA (High Level Architecture) platform.

HLA provider has same program layers as other providers and is based on the provider interface specification of CAO (Figure 1-1). Whereas other providers are designed to connect with FA devices, HLA provider is a gateway-type provider that just connects to HLA platform, and offers general-purpose functions that do not depend on specific application.

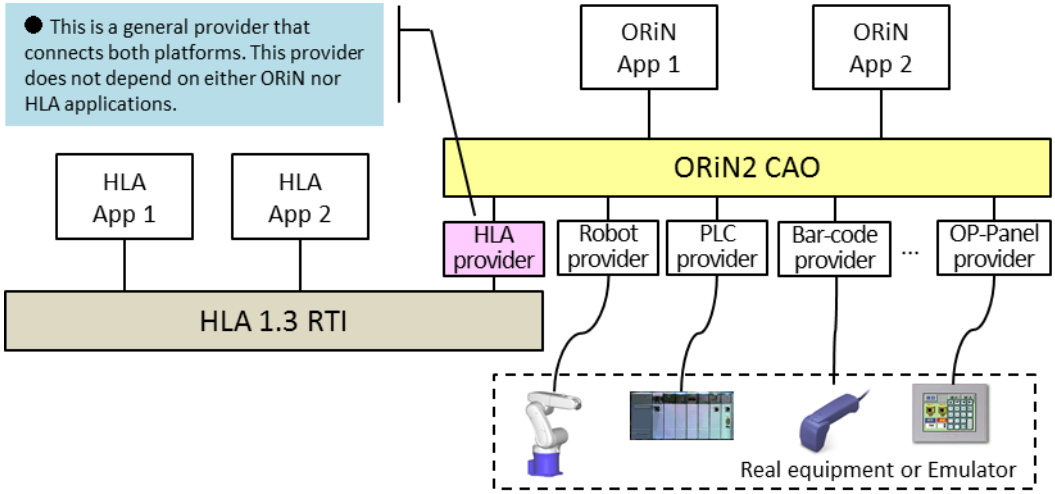

Figure 1-1: Role of HLA provider

HLA provider realizes seamless information exchange between CAO application and HLA application. For example, an HLA-connected simulator can easily acquire/configure information of actual FA devices via CAO. On the other hand, CAO application can easily acquire/configure information of the simulator via HLA. This achieves to combine virtual images and physical objects in the real world easily, allowing to develop more advanced application.

HLA provider, as explained just before, can be described as "a program that realizes free information exchange with HLA application", but also can be said as "a program that enables ORiN applications to use HLA functions". That is, for example, the message exchange with logical clock will be available not only to interaction between HLA applications but to interaction between ORiN applications. Thus, HLA provider can be said as a program that extends the CAO Engine function.

This documents describes the HLA provider's overview, how to use, and sample programs.

Note : To use HLA provider, you need to install HLA 1.3. RTI.
You need to register the provider registry after installing driver. For the way of registration, refer to Table 2-1.

2. Overview of HLA provider

2.1. Design Principles of Gateway Provider

While it can be said that the design principles of gateway provider is "to cover all HLA messages with Command class (CaoCommand) and Message class (CaoMessage)"; however, based on this notion, users may not feel advantages of provider so much because, when it comes to a client programs, they require two interfaces to use HLA and CAO functions, resulting in increase of overhead. The primary purpose of HLA provider development is that, based on the idea of CAO object model, "to manage HLA-based processes within HLA provider automatically so that applications that use HLA provider are unaware of these processes as much as possible."

For example, HLA has the notion of "ownership" which is given to only one attribute at the same time. An HLA object consists of attributes, and an attribute can be written only when the attribute possesses ownership. The notion of "ownership" does not exist in Object-oriented model as well as CAO. This might be one of the solution to reduce communication volume though, for client program-side, this results in increase of the number of processes. Since HLA provider handles the ownership management by itself, clients have no need to be aware of ownership. As a result, HLA provider allows users to use HLA functions easier than when they use HLA .

We will explain the internal process of HLA provider in Chapter 4.

2.2. Overview

HLA provider maps HLA objects to CAO objects. Viewed from CAO, HLA's Federates look like one controller, whereas viewed from HLA, CAO applications look like one Federate (see Figure 2-1). So, HLA provider is explained as a provider that connects two different platforms, not the one that is designed to a specific application. Therefore, object-relational mapping is designed for general-purpose, and the provider has a management function (HLAManager) to absorb the difference.

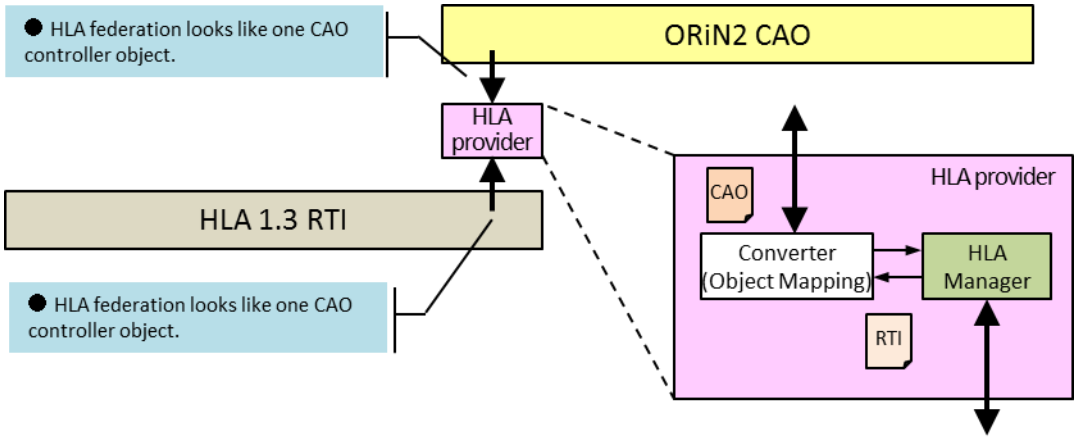

Figure 2-1: HLA provider

HLA's function is largely classified as follows.

- Federation Management
- Declaration Management
- Object Management
- Ownership Management
- Time Management
- Data Distribution Management[1]

Using HLA provider allows CAO application to use these functions. Regarding the way of using functions, refer to the following section. Regarding concrete example of these functions in provider, refer to Chapter 4.

[1] This function is not supported in ver.1.0.

2.3. Setup

The following shows the way of HLA provider setup.

(1) Construct HLA Runtime infrastructure

(2) Register HLA provider

To use HLA provider, you need to prepare Runtime infrastructure (RTI) of HLA 1.3. Behavior of this provider was checked by RTI of Pitch AB (http://www. pitch. se/). You can download the limited edition of RTI from the website.

Table 2-1: HLA provider

File name	CaoProvHLA. dll
ProgID	CaoProv. HLA
Registry registration[2]	regsvr32 CaoProvHLA. dll
Delete registry registration	regsvr32 /u CaoProvHLA. dll

[2] To register provider, use regsvr32.exe, or RegCOM.exe. ([Start]-[ORiN2]-[Tools]). Note that HLA provider cannot be registered unless HLA1.3.RTI is installed

2.4. Method and Property

2.4.1. CaoWorkspace::AddController method

Once CAO application calls AddController method of CaoWorkspace class, an HLA federate is created in HLA provider. That is, a controller object (CaoController) returned by AddController corresponds with an HLA federate.

The following shows argument specification of AddController method.

```
AddController
(
    "<Controller name>",            // Controller name
    "CaoProv. HLA",                 // Provider name. Fixed.
    "<Computer name>",              // Computer name where provider runs.
    "<Option>"            // Option character string
)
```

For < Provider name> , enter "CaoProv. HLA" at any time. For <Computer name>, enter a computer name where provider runs. This is a computer name where RTI of HLA runs as well. If both provider and client application run on the same computer, remain this argument empty.

To enter <Controller name>, follow the syntax below[3].

<Federation Execution Name>[. <Federate Type>] ・・・ (1)

Note that <Federation Execution Name> is used as the first argument of createFederationExecution() function of RTI and <Federate Type> is used as the first argument of joinFederationExecution() function of RTI. <Federate Type> is omittable. If this is omitted, a name entered in <Federation Execution Name> will be used to <Federate Type> as well.

In <Option> argument, you can enter an option listed in Table 2-2..

[3]You can enter any character string for <Controller name>. In this case, specify the controller name with this syntax by using Fed option (Table 2-2)

Table 2-2: Option character string of AddController

Option	Description
Url=<Fed file name>	Enter a URL of a file that contains FOM (Federation Object Model). [Mandatory]
Fed[=<Federation name>]	Specify the Federation execution name and Federate type with the syntax of (1). You can omit this entry if these items are entered in <Controller name>
TimePoli[=<Time Management policy>]	Specify Time Management policy. 0 – neither REGULATING nor CONSTRAINED 1 – REGULATING 2 - CONSTRAINED 3 – REGUALTING and CONSTRAINED (default : 0)
TimeAdv[=<Time advance>]	Specify how the time advances 0 – Time Step Advance (TAR) 1 – Event-based Advance (NER) 2 – Optimistic Advance (flushQueue) (default: 0)
CurTime[=<Current time>]	Initial value for logical time of Federate. (default : 0. 0) (Note) Available only when TimePoli is 1 or 3.
Lookahead[=<LookAhead value>]	Initial value for LookAhead of Federate. (default : 0. 5) (Note) Available only when TimePoli is 1 or 3.
AsyncDeli[=<Asynchronous delivery SW>]	Set valid or invalid the asynchronous delivery function of RO (Receive Order) message. 0 – Invalid 1 – Valid (default : 0)

The following shows an example of how to execute AddController method.

```
AddController
(
      "ChatRoom. Chat",
      "CaoProv. HLA",
      "",
      "Url=file:///D:/ORiN2/CAO/ProviderLib/HLA/Bin/Chat. fed, TimePoli=1, CurTime=2"
);
```

2.4.2. CaoController::AddVariable method

Since an object of HLA is mapped to a Variable object of CAO, to operate an HLA object, create a Variable object by using AddVariable method first. Then, operate InteractionClass and ObjectClass by specifying variable name and/or option character string. These classes are registered in RTI at the creation of variable objects and are eliminated from RTI when they are deleted.

The following shows arguments specification of AddVariable method.

```
CaoController::AddVariable
(
    BSTR bstrName,                  // Variable name
    BSTR bstrOption                 // Option
);
```

To enter <Variable name>, follow the syntax below..

<FOM object> [# < data type >]

For <FOM object>, enter a parameter name of InteractionClass or an attribute of ObjectClass with full path. Use "." (period) for delimiter. Based on the specification of FOM, this must start with "InteractionRoot" or "ObjectRoot". To enter an ObjectClass, specify the attribute of Object with "."(period) delimiters, and then type "!"(exclamation mark) and enter an instance name (object name).

For <data type>, specify a parameter or data type of an attribute.

Table 2-3 shows the data types that can be specified from HLA provider. If data type is not specified, it will be treated as character string (VT_BSTR). For example, if " InteractionRoot. Communication. ABC#2" is entered, HLA provider treats "ABC" as 2-byte signed Integer (VT_I2). In HLA specification, data marshaling is assigned to federates, so HLA provider performs marshaling in place of CAO application. Therefore, it is very important to specify data type exactly. Especially, when the value is Integer, you need to adjust memory storage type by using option (4) Big endian conversion; otherwise HLA provider may fail to read/write data from/to RTI correctly.

Table 2-3: Data type available in HLA Provider

Data type	Value*1	Byte	Description
VT_I1		1	One-byte signed Integer
VT_UI1	17	1	One-byte unsigned Integer
VT_I2	2	2	Two-byte signed Integer
VT_UI2		2	Two-byte unsigned Integer
VT_I4	3	4	Four-byte signed Integer
VT_UI4		4	Four-byte unsigned Integer
VT_R4	4	4	Single-precision floating-point number (32-bit)
VT_R8	5	8	Double-precision floating-point number (64-bit)
VT_BSTR	8	Variable	Character string Specify the number of characters in VT_UI4 first, and then enter UNICODE characters with NULL terminator.
VT_BOOL	11	1	TRUE (-1), FALSE (0)

*1 : Enter these values to specify respective data type on the left column.

With the following <Option>, you can specify the behavior of HLA provider.

Option=<Option value>

To specify <Option value>, use an Integer. Table 2-4 shows the option list. You can specify multiple options by specifying the sum of these option values.

Table 2-4: Options in Variable class

Value	Option name	Target	Description
1	Event invalidation	Interaction	In the default setting, Interaction message is issued by OnMessage event of ControllerClass. When this option is specified, Interaction message is stored in the queue instead, and no event is issued. The stored value can be retrieved by Value property.
2	Object creation	Object	Create a specified HLA object at the time of an Variable object creation. When this option is invalid and the specified HLA object does not exist at that timing, it fails to create a variable object.
4	Big endian conversion	Interaction, Object	Convert data to Big Endian at data reading/writing. Use this option to exchange data between different memory storage types.

8	Prohibit to release Ownership	Object	In the default setting, a Federate gives the ownership of Object when it is asked by other Federate. If this option is specified, a Federate denies to give the ownership to others.
16	Unowned ownership pull	Object	In the default setting, once the writing of a variable object occurs, the variable object gets the ownership (Intrusive Pull). If this option is specified, an object can get the ownership only when no objects do not obtain it.
32	Publish not declared	Interaction, Object	In the default setting, when a variable object is created, it declares both Publish and Subscribe. If this option is specified, a variable object declares Subscribe only. Note that this option is ignored if (2) Object creation option is set.
64	Subscribe not declared	Interaction, Object	In the default setting, when a variable object is created, it declares both Publish and Subscribe. If this option is specified, a variable object declares Publish only.

The following shows the sample programs of AddVariable method execution.

[For InteractionClass]

```
AddVariable
(
    " InteractionRoot. Communication. Message ",
    " Option=32"                              // Do not declare as "Publish"
);
```

[For ObjectClass]

```
AddVariable
(
    " ObjectRoot. myClass. myAttr!XYZ "      // XYZ is an object name
    " Option=2 "                              // Create a new object
);
```

2.4.3. CaoController::Execute method

The following table shows commands available in Execute method.

Table 2-5: Execute command

Command	Argument	Description
startTimer	none	Validate the thread for event processing *1
stopTimer	none	Invalidate the thread for event processing

*1: HLA provider internally processes the event processing thread that runs HLA interaction and

synchronous points. This thread becomes effective at the timing of AddController method execution. In general, processes that run in this thread will be practical after when interaction and/or attributes required by applications are added by AddVariable method or other ways. To suspend this thread, use "stopTimer" command. To resume it, use "startTimer" command.

2.4.4. CaoVariable::get_Value property

If CAO variable object is an ObjectClass of HLA, get_Value obtains the current value of ObjectClass. If CAO variable object is InteractionClass of HLA and 11) Event invalidation, which is a CAO Variable class option, is specified, get_Value obtains Interaction messages stored in the queue.

2.4.5. CaoVariable::put_Value property

If CAO variable object is ObjectClass of HLA, put_Value sets the current value of an attribute which corresponds to the CAO object. If CAO variable object is InteractionClass of HLA, put_Value sends an Interaction message. In this case, an ID of Controller object which has CAO variable object collection is embedded to the tag of the Interaction message. This enables the receiver of this message to find out the controller object where the received message belongs.

2.4.6. CaoVariable::get_ID property

If CAO variable object is InteractionClass of HLA, ID property stores an ID that has been given automatically by HLA provider.

2.4.7. CaoVariable::get_Help property

If CAO variable object is InteractionClass of HLA, Help property stores tag information of the received Interaction message. If a message has been sent from CAO application, controller ID of CAO Controller object[4], which sent the message is stored in get_Help.

[4] For Ver.1.1.2 or lower, CAO Variable object ID is stored.

2.5. Variable list

2.5.1. Controller class

As explained in the previous section, Interaction and Object defined in FOM of HLA are mapped to CAO Variable class. There are some CAO Variable classes, such as Controller class, Robot class. Among them, HLA provider uses variables of Controller class.

CAO variable has two types; user variable and system variable. HLA provider assigns FOM defined-Interaction and/or Object to user variables of Controller class and also assigns system variables that controls HLA behavior to system variables. The following table shows the details.

Table 2-6: Controller class User variable list

Variable name	Data type	Description	Attribute	
			get	put
<any Interaction>	<Dat type in Table 2-3>	Treat an Interaction defined in FOM as a variable	✓	✓
<any Object>	<Data type in Table 2-3>	Treat an Object defined in FOM as a variable.	✓	✓

Table 2-7: Controller class System variable list

Variable name	Data type	Description	Attribute	
			get	put
@CURRENT_TIME	VT_R8	Current time is adjusted by writing the logical current time and value of Federate.	✓	✓
@ADVANCE_TIME	VT_R8	Advance the logical current time of Federate by specified time period.	-	✓
@LOOKAHEAD	VT_R8	LookAhead time period of Federate	✓	✓
@LBTS	VT_R8	LBTS (Lower Bound Time Stamp) time.	✓	-
@NEXTEVENT_TIME	VT_R8	TSO event time on the top (Minimum Next Event Time).	✓	-
@SYNC_POINT	VT_BSTR	Set the synchronization point of Federation. Once this value is specified, OnMessage(1) event occurs.	-	✓

2.6. Event

2.6.1. CaoController::OnMessage event

As explained in 2.1 Design Principle, HLA provider is not designed to simply wrap all HLA events but is designed to control all events and encapsulate the minimum events required by client in CaoMessage object and then hand it with OnMessage event. The type of event can be distinguished by Number property of CaoMessage object. (Table 2-8)

Table 2-8: Message list of OnMessage event

No.	Message			Response	
Number	Value	Destination	Source	Clear	Reply
1	Synchronous message [5]	<Empty>	<Empty>	Synchronization completion notification	-
2	<Empty>[6]	<Empty>	<Empty>	-	-
3	Interaction message [7]	ID of the received variable object	ID of the source controller object	-	Interaction message sending[8]

2.7. Error code

CRD provider does not have original error code. For about ORiN2 common errors, refer to the error code of "ORiN2 Programming Guide.

[5] This event occurs if Synchronization point notification (announceSynchronizationPoint) is arrived from HLA.
[6] This event occurs if Synchronization confirmation notification (federationSynchronized) is arrived from HLA.
[7] This event occurs if Interaction message is arrived (receiveInteraction) from HLA. However, if (1) Event invalidation is set in AddVariable option, this event does not occur.
[8] This can save and respond the context of Interaction message of HLA.

3. Sample program

This chapter shows the sample program written in Visual Basic and the execution result.

List 3-1 Sample. frm

```
Private caoEng As CaoEngine
Private caoWS As CaoWorkspace
Private WithEvents caoCtrl As CaoController
Private caoVar As CaoVariable

Private Sub Form_Load()

    Set caoEng = New CaoEngine
    Set caoWS = caoEng. Workspaces(0)

    ' Create Federate
    Set caoCtrl = caoWS. AddController("ChatRoom. Chat", "CaoProv. DNWA. HLA", "", _
                  "URL=file:///D:/ORiN2/CAO/ProviderLib/HLA/Bin/Chat. fed")
    Set caoVar = caoCtrl. AddVariable("InteractionRoot. Communication. Message", "")

End Sub

Private Sub Command1_Click()

    caoVar. Value = Text1. Text

End Sub

Private Sub caoCtrl_OnMessage(ByVal pICaoMess As CAOLib. ICaoMessage)

    List1. AddItem pICaoMess. Value

End Sub
```

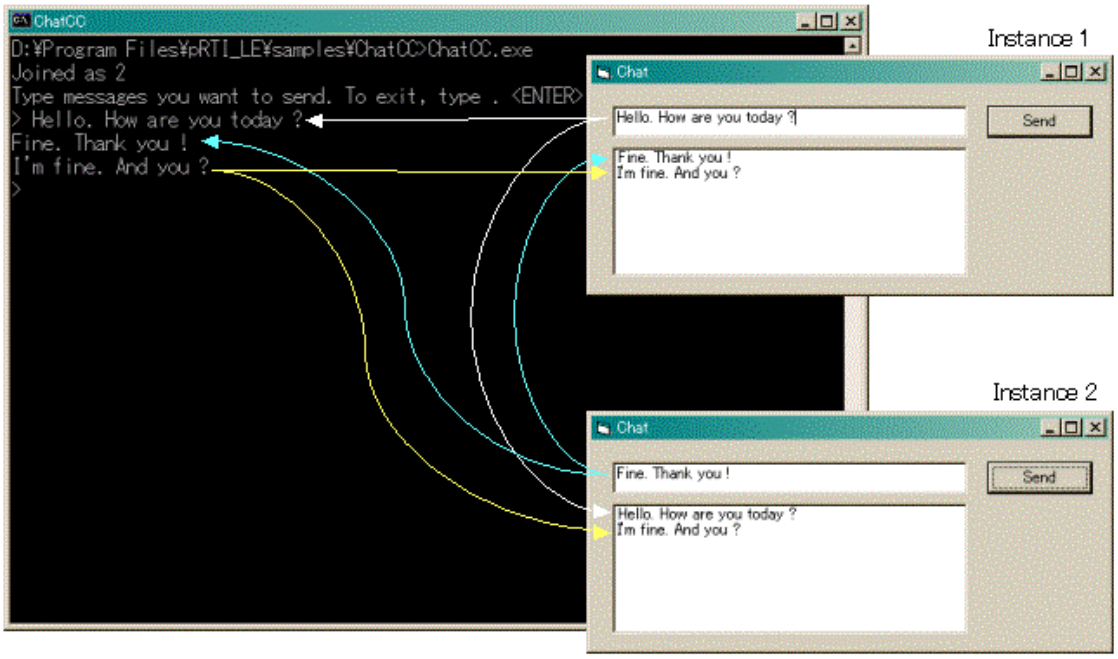

Figure 3-1: Execution result of the sample program

This sample program runs as the same manners as "ChatCC" sample program that comes with pRTI Limited Edition of Pitch AB (http://www. pitch. se/). Comparing this program and ChatCC program, you may be able to understand the mechanism of HLA provider more deeply. As Figure 3-1 shows, an HLA message can be exchanged between this program and ChatCC program as well as between this programs.

[Reference]

To better understanding of whole functions of HLA provider, another test program is prepared (Figure 3-2). With reference to this program source code, you can understand how to use HLA provider more deeply.

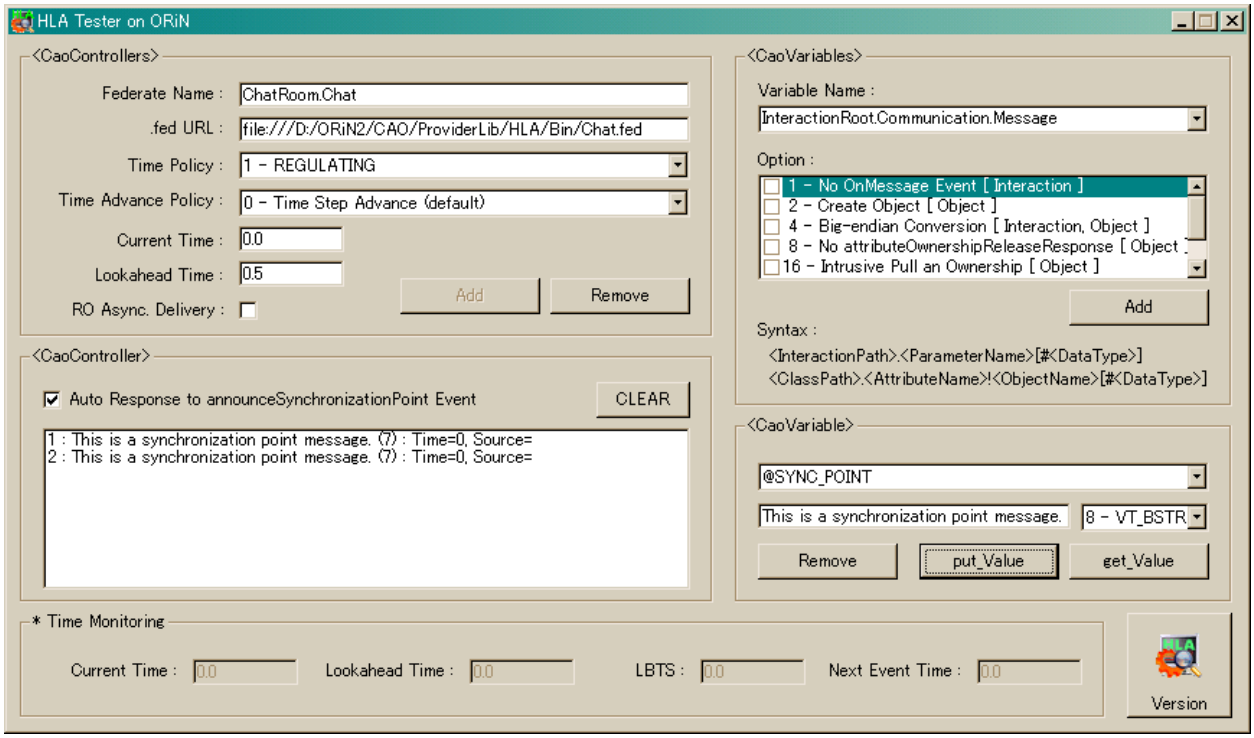

Figure 3-2: Test program

4. Internal operation mechanism

Previous chapters focus on how to use HLA provider from the client (CAO application) side. To understand the provider more deeply, this chapter explains how HLA functions are used in HLA provider inside.

HLA's functions are roughly classified into the following categories.

- Federation Management
- Declaration Management
- Object Management
- Ownership Management
- Time Management
- Data Distribution Management

Here, the Data Distribution Management function works merely to reduce the communication volume; therefore, version 1.0 does not use this function. The following section briefly explains the timing when other functions are used.

4.1. Federation management

This function manages Federation entirely, such as managing each Federate, message routing, and synchronization management. These functions are mapped to Controller object of CAO (CaoController). That is, a CaoController object corresponds to a Federate of HLA.

Concretely, when AddController method of CaoWorkspace class is called, a Federation is created in HLA provider. If the Federation has been created already, CaoController object joins the Federation. On the other hand, when CaoController object disappears, it resigns from the Federation (Figure 4-1). Since the HLA management function closes in this timing, it is impossible to communicate with HLA once CaoController object disappears.

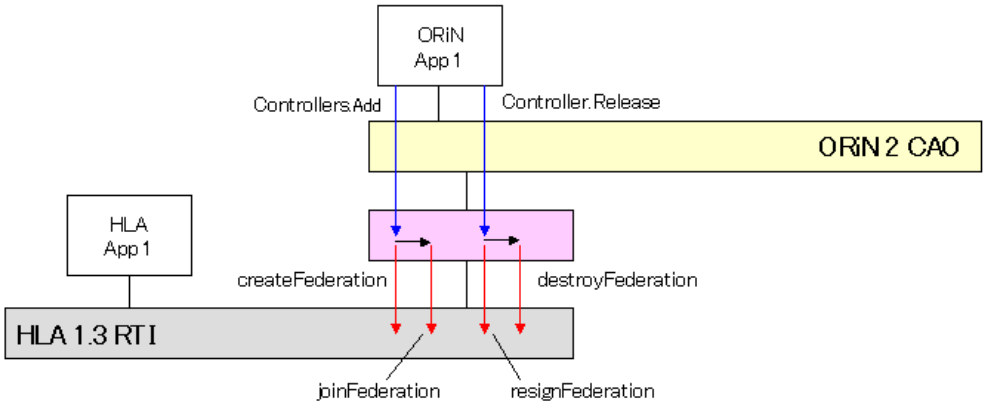

Figure 4-1: Federate life cycle

HLA has a function that synchronizes each Federate instantly (Synchronization Point). This function enables

CAO application to synchronize with HLA application as well (Figure 4-2). Of course, this function enables to synchronize two CAO applications.

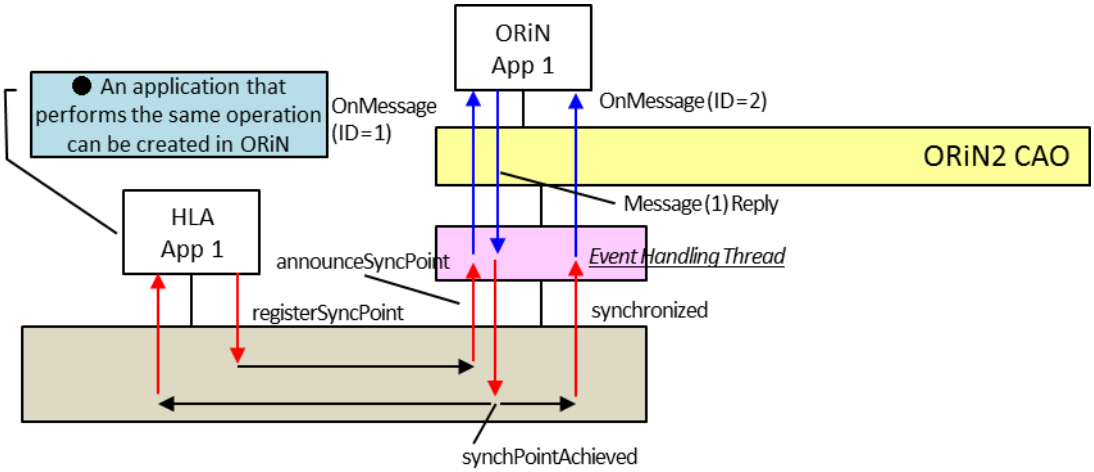

Figure 4-2: Federate synchronization

4.2. Declaration management

This function manages declaration of Interaction or Class object (Object); for example, declaration of publisher/subscriber model. When using HLA Interaction and/or Object, each Federate must declare correctly whether if the Federate publishes Interaction/Object class (that is; the Federate creates an Interaction/Object and update the value) or it subscribes Interaction/Object (that is; the Federate refers an Interaction/Object only).

This function does not exist in Object-oriented model. This might be one of the solution to reduce the communication volume though, for client program-side, this result in increase of processes. Therefore, HLA provider internally performs this declaration automatically.

Concretely, the declaration is made when AddVariable method of CaoController class is called, in other word, when a variable object is created. In the default setting, both publish and subscribe are declared (Figure 4-3, Figure 4-4). If Value property of variable object is "always written only" or "always read only", this declaration can be made by AddVariable option (Table 2-4). Table 4-1 shows the attribute values (Attribute property) of CAO variable object.

HLA object can make this declaration in each attribute. To declare different attribute of an object which has already been created, it is necessary to recreate AHL (Attribute Handle Set) and make a declaration again. Because HLA provider allocates one attribute to one variable object, this might happen at any time. Such dynamic re-declaration is automatically processed inside of HLA provider as well.

Table 4-1: Attribute value of CAO variable object

Declaration of HLA	Attribution value of variable
Subscribe	1 – Readable
Publish	2 – Writable
Subscribe & Publish	3 – Readable/Writable

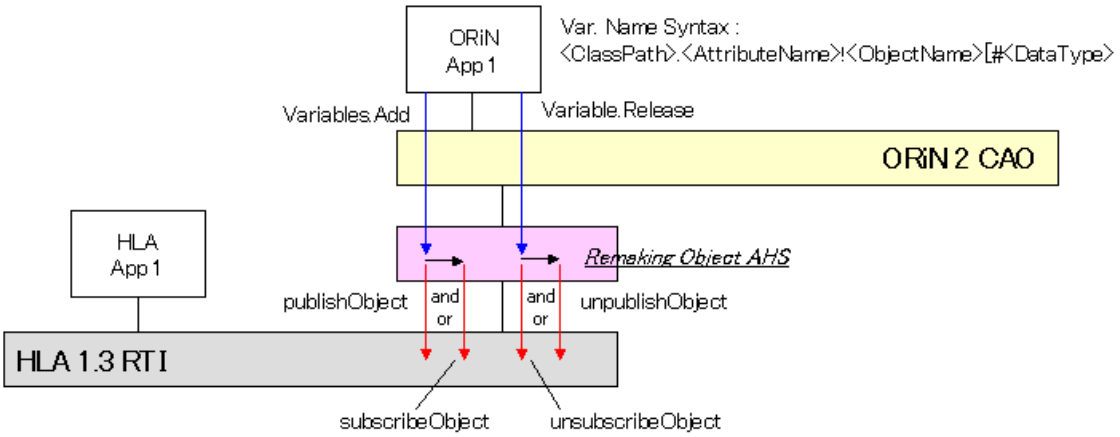

Figure 4-3: Declaration of an Object

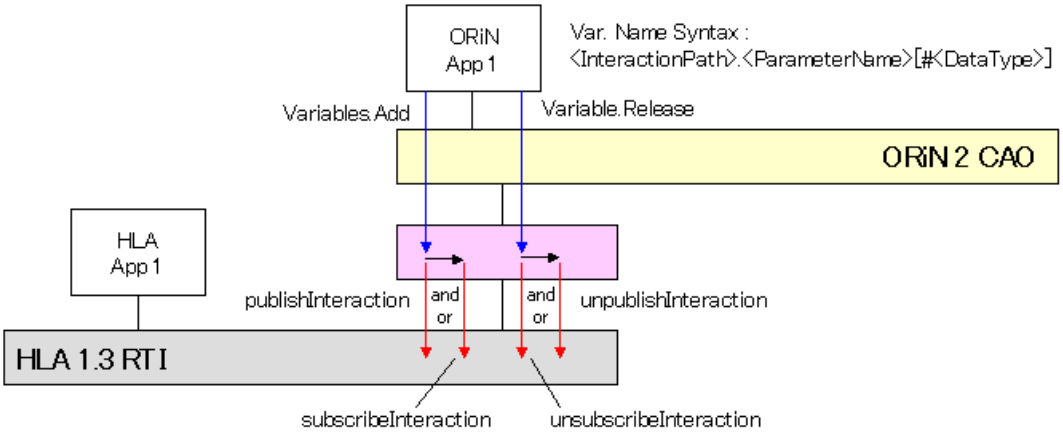

Figure 4-4: Declaration of an Interaction

4.3. Object management

This function manages the lifecycle, update, and reference of Interaction and Class object. As explained earlier, since HLA's Interaction and Class object are mapped to Variable objects of CAO, functions explained here correspond with the creation, deletion, value reading/writing of CAO Variable objects. (Figure 4-5).

Since attribute values of HLA objects are cached in HLA provider, CAO application can refer these values at any time.

It seems that the mechanism of HLA's data exchange is to transfer memory image only, so the marshaling process must be done by Federate. HLA provider performs marshaling automatically based on the data type and the memory storage type. However, to perform marshaling correctly, be sure to specify the data type and the memory storage type at the execution of AddVariable method. To specify the data type and the memory storage type, use a variable type and an option, respectively.

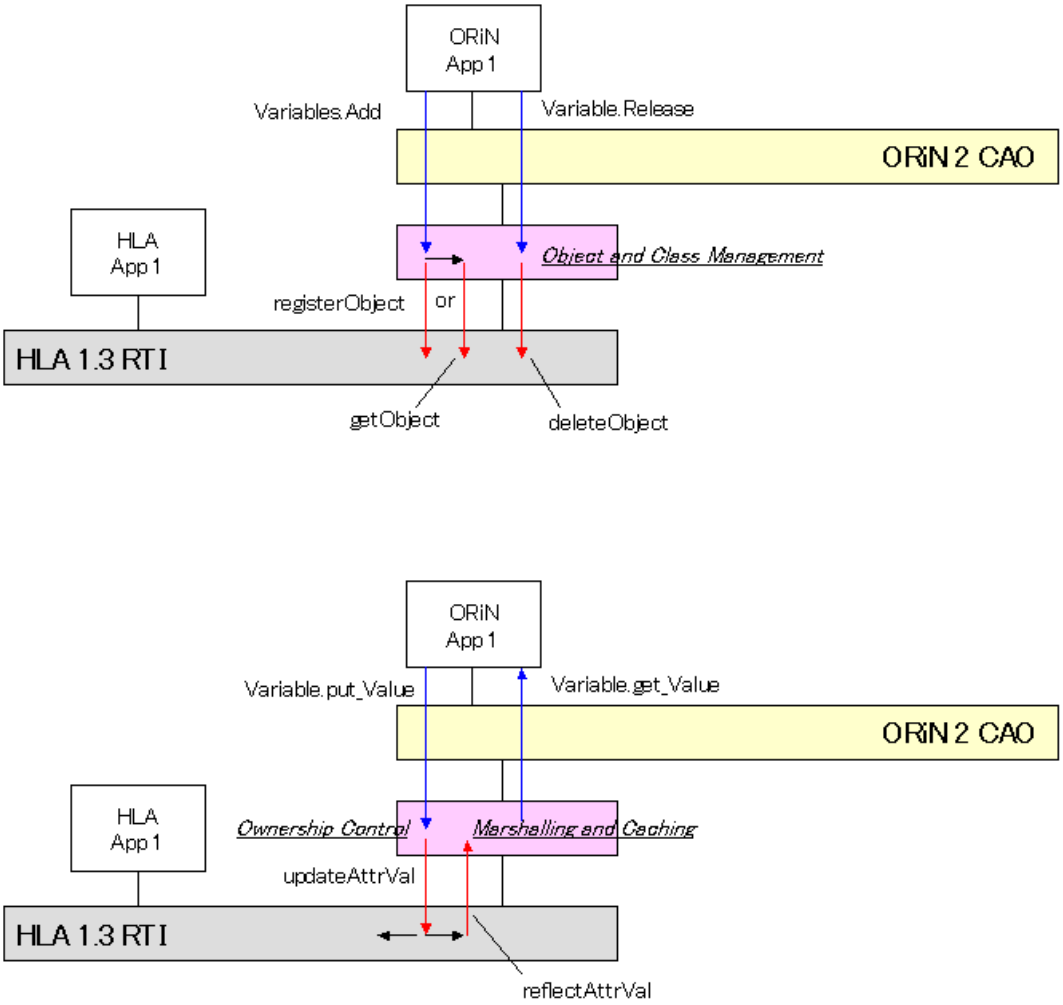

Figure 4-5: Update and reference of object attribute

In default setting, HLA's Interaction is not cached (queued). Once an Interaction is received, OnMessage

event is issued (Figure 4-6). To change the behavior so that it queues Interactions, select (1) Event invalidation of AddVariable option.

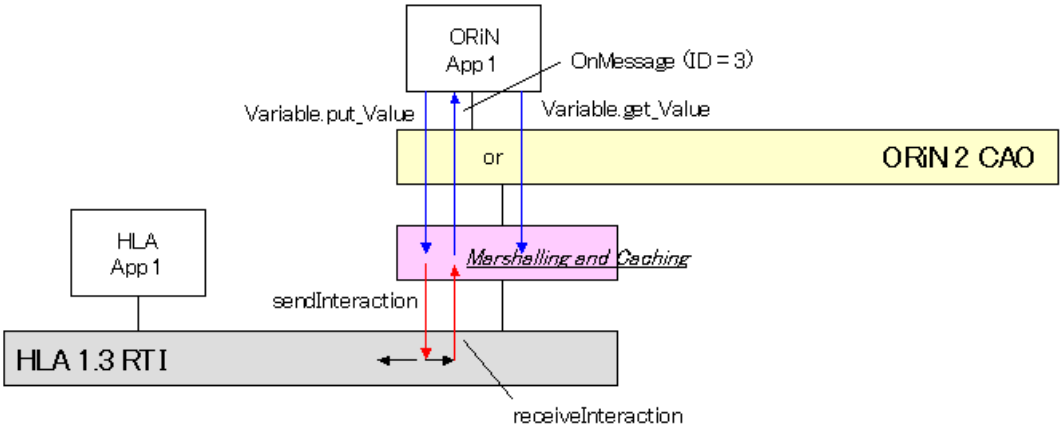

Figure 4-6: Send and Receive of Interaction parameters

When an HLA Interaction that is mapped to an CAO Variable (CaoVariable) object issues a message, an ID of Controller object that includes Variable object collection is automatically added to a tag of the message. The ID is stored in the Source property of CAO message (CaoMessage) object that stores the message (Figure 4-7). This allows receivers to find out where the message comes from, so, it is practical when one Interaction is shared with several Federates.

Also, once Reply method of CAO Message object is executed, put_Value of CAO Variable object that receive a message is called and each Federate can receive the Reply message as well.

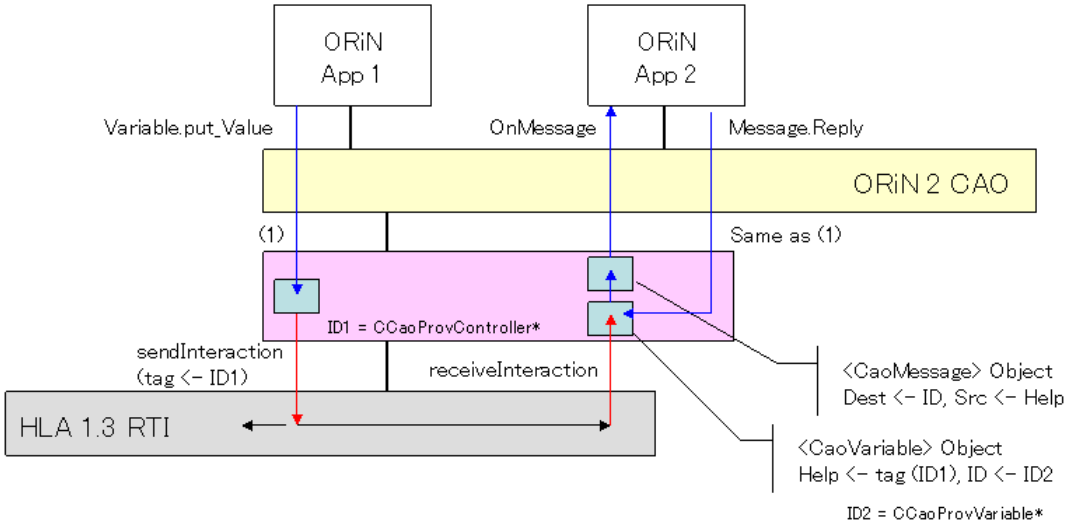

Figure 4-7: Interaction Sharing

4.4. Ownership Management

This function manages ownership of Class object. Ownership is managed in respective attribute of object. To update an attribute value, the attribute must obtain ownership. Since only one ownership exists in a Federation, only one Federate can be updated at the same time. A Federate can acquire ownership by negotiating with other Federate and, in reverse, can release ownership.

The notion of "ownership" does not exist in Object-oriented model as well as CAO. This might be one of the solution to reduce communication volume though, for client program-side, this results in increase of the number of processes. Since HLA provider handles the ownership management by itself, clients have no need to be aware of ownership.

Concretely, if own Federate does not have ownership at the timing of variable writing, HLA provider retrieves ownership and writes the value. In the default setting, the ownership acquisition policy is "Intrusive Pull". If AddVariable option (16) "Unowned ownership pull" is set, a Federate can acquire ownership only when no Federate obtains ownership. If it fails to acquire ownership, an error is returned.

On the other hand, if own Federate is requested to release ownership by another Federate, it releases ownership. This release will be prohibited if option (8) Prohibit to release Ownership is set. In this case, a Federate that requests the release of ownership keeps requesting ownership but it fails; as a result, as time goes by, ownership gradually focus on a Federate created by HLA provider.

4.5. Time Management

This is the most significant function of HLA. This function coordinates logical time of respective Federate. All Federate can have own logical time independently. Note that there is no universal time for entire Federation.

In this Time Management, there are several elements regarding to logical time illustrated in Figure 4-8: Time ManagementFigure 4-8. These items are mapped to Variable object (CaoVariable) of CAO. To advance the logical time, specify value in the Variable object.

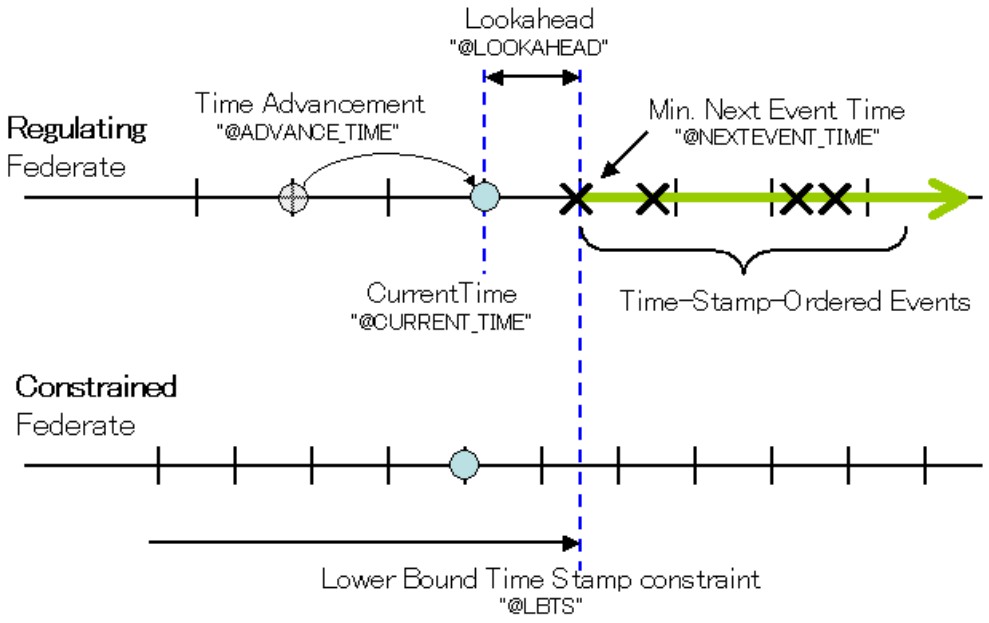

Figure 4-8: Time Management

All Federate has one of the Time Management policy from the following four types.

(1) neither REGULATING nor CONSTRAINED
(2) REGULATING
(3) CONSTRAINED
(4) REGUALTING and CONSTRAINED

Among them, policy (1) is not restricted at all regarding to time. Only "REGULATING" Federate can issue an TSO (Time Stamp Order) event. Concretely to say, for example, this Federate can send an Interaction parameter with logical time and update Object attribute.

CAO application can create Federates of these four types. For example, when CAO application creates a CaoController object, it specifies one of these four types with AddController option (TimePoli) (see Figure 4-9). To advance time, specify the value of @CURRENT_TIME or @ADVANCE_TIME of system variables.

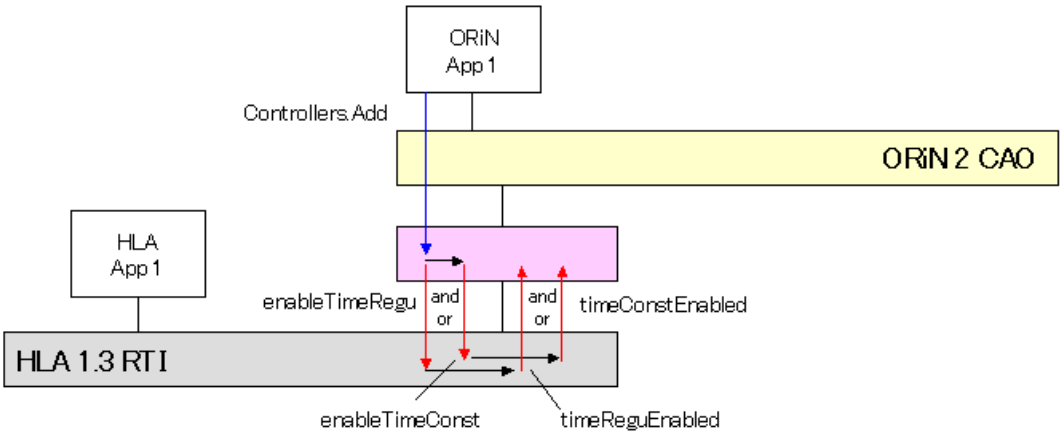

Figure 4-9: Time Management policy

There are three ways to advance the time of HLA. TSO event to be obtained differs depending on the way of time advance.

(1) Time Step Advance (TAR)

(2) Event-based Advance (NER)

(3) Optimistic Advance (flushQueue)

To specify how to advance the logical time, use AddController's option (TimeAdv). Since Time Management policy and how to advance the logical time are decided at the time of CaoController object creation, it cannot be change ever after. To change these settings, you need to re-create CaoController object.

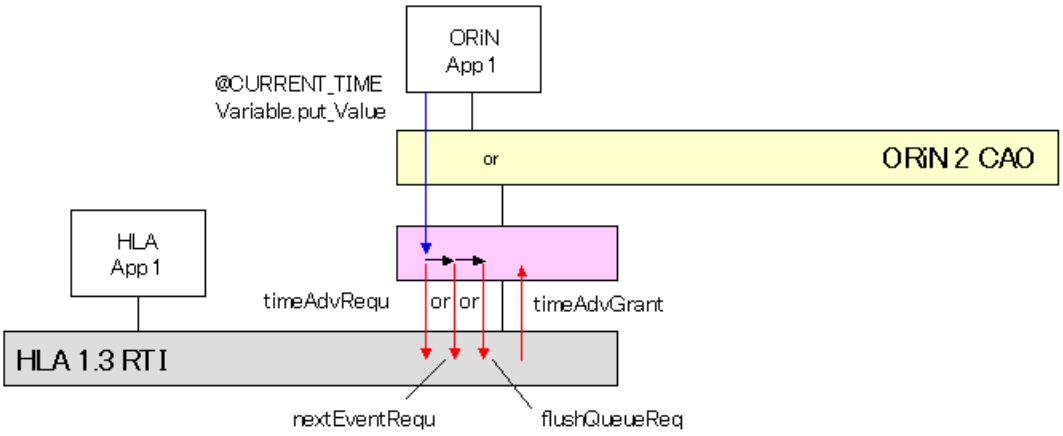

Figure 4-10: How to advance Logical time