HLA プロバイダ

HLA 用ゲートウェイ

Version 1.0.0

ユーザーズ ガイド

April 11, 2022

【備考】

本プロバイダは, (財)機械振興協会 技術研究所が競輪の補助により実施した研究成果の一部を使用しています.

【改版履歴】

バージョン	日付	内容
1.0.0.0	2006-02-24	初版.
1.0.0.1	2010-02-11	エラーコード追加
1.0.0.2	2011-03-11	プロバイダ登録ツールに関する追記
1.0.0	2012-07-17	ドキュメントのバージョンルールを変更
	2022-04-11	ユーザーズガイドを修正しました.

【対応機器】

機種	バージョン	注意事項
Pitch AB	1.3	Pitch AB (http://www.pitch.se/) の pRTI 1.3 で動作確認.

目次

1. はじめに

本書はORiN2 CAO(Controller Access Object)とHLA(High Level Architecture)とのプラットフォーム間接続のためのゲートウェイプロバイダ(以後 HLA プロバイダと呼ぶ)のユーザーズガイドです.

HLA プロバイダはその他のプロバイダと同じレイヤのプログラムで, CAO のプロバイダ・インタフェース仕様に基づいています(図 1-1). 他のプロバイダが FA 機器に接続するためのプロバイダであるのに対して, HLA プロバイダは HLA プラットフォームと接続するためのゲートウェイタイプのプロバイダで, 特定のアプリケーションに依存しない汎用的な機能を与えます.

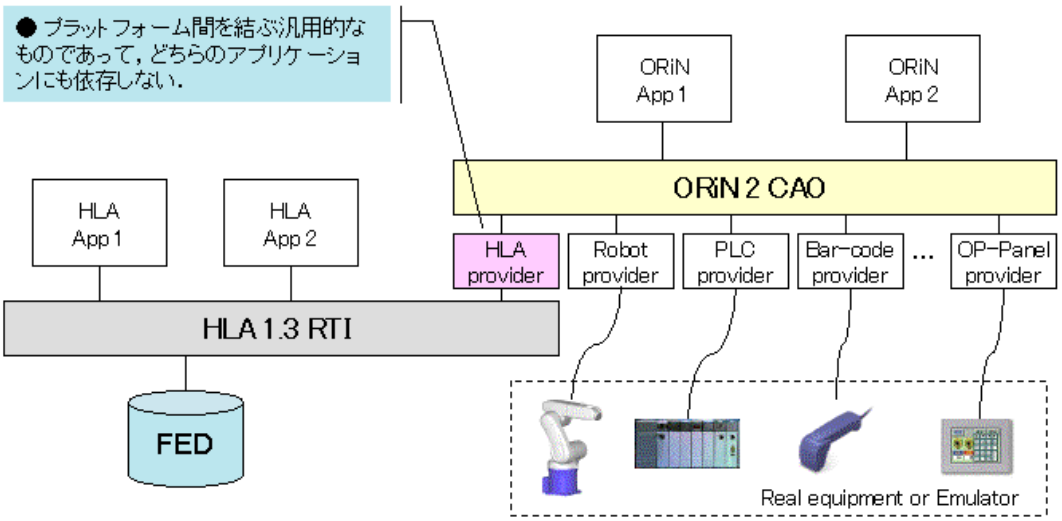

図 1-1: HLA プロバイダの役割

この HLA プロバイダにより, CAO アプリケーションと HLA アプリケーションはシームレスに情報交換することができるようになります. 例えば, HLA に接続されているシミュレータは CAO を介して FA 機器などの実機の情報を容易に取得・設定することができ, 逆に CAO アプリケーションは HLA を介してシミュレータの情報を容易に取得・設定できるようになります. これにより, 現実世界とシミュレーション世界を簡単に融合でき, 新しいアプリケーションの開発が期待できます.

このように, HLA プロバイダは「HLA アプリケーションとの自由な情報交換を実現するプログラム」ですが, 「HLA の機能を ORiN アプリケーションから使えるようにするプログラム」という見方もできます. すなわち, HLA アプリケーションとのインタラクションではなくて, ORiN アプリケーション同士のインタラクションでも, 例えば論理時刻付きメッセージの授受が可能になるのです. 従って, HLA プロバイダは CAO エンジンの機能を拡張するプログラムとも言えるのです.

本書では, この HLA プロバイダの概要や使い方, サンプルプログラムなどを解説します.

注意: HLA プロバイダを使用するには, HLA 1.3 RTI をインストールしなければなりません.
ドライバインストール後にプロバイダをレジストリ登録する必要があります. レジストリ登録の方法は表 2-1 を参照してください.

2. プロバイダの概要

2.1. 設計方針

ゲートウェイプロバイダの設計方針としては，「HLA のすべてのメッセージを CAO のコマンドクラス(CaoCommand)やメッセージクラス(CaoMessage)を用いてラップする．」という方針も考えられますが，これではクライアントプログラムからするとCAOとHLAの機能をそれぞれのインタフェースを用いて使うのと大差なく，返ってオーバーヘッドが増えるだけで余りメリットが感じられません．HLA プロバイダはそうではなくて，「CAO のオブジェクトモデルに従って，そこにない HLA の概念はプロバイダの中で自動的に管理し，HLA プロバイダを使うアプリケーションには出来る限りそれを意識させないようにする．」という方針で開発しました．

例えば，HLA のオブジェクトはアトリビュートごとに所有権という考えがあり，所有権がないと書き込みができません．CAO にはそのようなアイデアはなく，そもそもいわゆるオブジェクト指向の世界にもないアイデアだと思われます．HLA の資料によるとこれは通信量を減らすための一つのアイデアであるとのことですが，クライアントプログラムからするとそれだけ手続きが増えてしまうのです．HLA プロバイダはこのような所有権の管理をプロバイダの内部で自動的に行うことによって，そのクライアントに所有権を意識させません．結果的に，HLA プロバイダを使うとネイティブに HLA を使うよりもより簡単に HLA の機能を使うことができるといえます．

HLA プロバイダの内部処理に関しては，4 章でもう少し詳しく述べます．

2.2. 概要

HLA プロバイダは CAO のオブジェクトと HLA のオブジェクトをマッピングし，CAO からみると HLA のフェデレーション(Federation)が一つのコントローラに見え，逆に HLA から見ると CAO のアプリケーションが一つのフェデレート(Federate)に見えるようにします(図 2-1)．このように，HLA プロバイダは単に特定のアプリケーション用のプロバイダではなく，二つのプラットフォームを結合するためのプロバイダであり，そのためにオブジェクトのマッピングも汎用的に設計され，その違いを埋めるための管理機能(HLA Manager)を有しています．

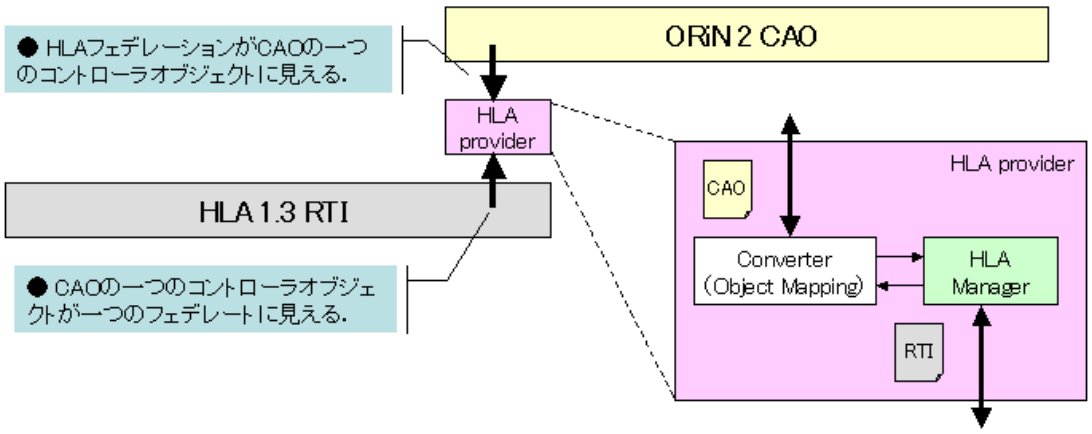

図 2-1： HLA プロバイダ

HLA の機能を大別すると，下記のようになります．

・フェデレーション管理 (Federation Management)
・デクラレーション管理 (Declaration Management)
・オブジェクト管理 (Object Management)
・所有権管理 (Ownership Management)
・時刻管理 (Time Management)
・データ分散管理 (Data Distribution Management)[1]

HLA プロバイダを使うことによって, これらの機能を CAO アプリケーションから使うことができます. その使い方は, 以下の節を参照して下さい. これらの機能がプロバイダの中でどのように使われているかは, 4 章を参照して下さい.

[1] ver.1.0 ではこの機能は使えません.

2.3. セットアップ

HLA プロバイダのセットアップ手順は下記のようになります.

(1) HLA のランタイム環境の構築

(2) プロバイダのレジストリ登録

HLA プロバイダを使うには HLA 1.3 のランタイム環境(RTI, Run Time Infrastructure)が必要です. 本プロバイダは Pitch AB (http://www.pitch.se/) の RTI で動作確認しております. このサイトから Limited Edition の RTI をダウンロードすることができます.

表 2-1: HLA プロバイダ

ファイル名	CaoProvHLA.dll
ProgID	CaoProv.HLA
レジストリ登録[2]	regsvr32 CaoProvHLA.dll
レジストリ登録の抹消	regsvr32 /u CaoProvHLA.dll

[2] プロバイダの登録は regsvr32.exe または RegCOM.exe ([スタート]→[ORiN2]→[Tools])で実行できます. HLA 1.3 RTI をインストールしていないと, HLA プロバイダの登録はできません.

2.4. メソッド・プロパティ

2.4.1. CaoWorkspace::AddController メソッド

CAO アプリケーションが CaoWorkspace クラスの AddController メソッドを呼び出したときに, HLA プロバイダ内部で HLA のフェデレートが作成されます. つまり AddController によって返されるコントローラオブジェクト(CaoController)が HLA のフェデレートに対応します.

AddController メソッドの引数仕様は下記のようになっています.

```
AddController
(
    "<コントローラ名>",          // コントローラ名
    "CaoProv.HLA",              // プロバイダ名. 固定.
    "<マシン名>",                // プロバイダ実行マシン名.
    "<オプション>"               // オプション文字列
)
```

ここで, <プロバイダ名>は固定で“CaoProv.HLA”を指定します. また, <プロバイダ実行マシン名>はプロバイダを稼動させるマシン名を指定します. 同時にそれは HLA の RTI が稼動しているマシン名でもあります. クライアントアプリケーションと同じマシンで実行する場合は空白を指定します.

<コントローラ名>は下記の文法で指定します[3].

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

ここで<Federation Execution Name>は RTI の createFederationExecution()関数の第 1 引数で, 同じく <Federate Type>は joinFederationExecution()関数の第 1 引数で用いられます. <Federate Type>は省略可能で省略した場合は<Federation Execution Name>と同じ名前を<Federate Type>に使います.

最後に<オプション>では, 表 2-2 のオプションを指定することができます.

[3]<コントローラ名>は任意の文字列を指定することもできます. その場合は, Fed オプション(表 2-2)を使って同じ文法で指定します.

表 2-2: AddController のオプション文字列

オプション	意味
Url=<Fed ファイル名>	FOM(Federation Object Model)を記述したファイルの URL を指定します.【必須】
Fed[=<フェデレーション名>]	フェデレーション実行名とフェデレートタイプを(1)の文法で指定します. コントローラ名で指定した場合は, 重複して指定する必要はありません.
TimePoli[=<時刻管理ポリシー>]	時刻管理ポリシーを指定します. 0 – neither REGULATING nor CONSTRAINED 1 – REGULATING 2 - CONSTRAINED 3 – REGUALTING and CONSTRAINED （デフォルト値: 0）
TimeAdv[=<時刻の進め方>]	時刻の進め方を指定します. 0 – Time Step Advance (TAR) 1 – Event-based Advance (NER) 2 – Optimistic Advance (flushQueue) （デフォルト値: 0）
CurTime[=<現在時刻>]	フェデレートの論理時刻の初期値. （デフォルト値: 0.0） (注) TimePoli=1 or 3 の時のみ有効.
Lookahead[=<LookAhead 時間>]	フェデレートの LookAhead 時間の初期値. （デフォルト値: 0.5） (注) TimePoli=1 or 3 の時のみ有効.
AsyncDeli[=<非同期配達 SW>]	RO(Receive Order)メッセージの非同期配達機能の有効・無効を指定します. 0 – 無効. 1 – 有効. （デフォルト値: 0）

以下に AddController メソッドを実行するときの例を示します.

```
AddController
(
        “ChatRoom.Chat”,
        “CaoProv.HLA”,
```

```
        “”,
        “Url=file:///D:/ORiN2/CAO/ProviderLib/HLA/Bin/Chat.fed, TimePoli=1, CurTime=2”
);
```

2.4.2. CaoController::AddVariable メソッド

HLA のオブジェクトは CAO の変数オブジェクトにマッピングされているので, HLA のオブジェクトを操作したい場合は AddVariable メソッドで変数オブジェクトを作成します. 作成するときの変数名やオプション文字列で InteractionClass や ObjectClass のオブジェクトを操作します. それらは, 変数オブジェクトの生成時に RTI に登録され, 消滅時に RTI から削除されます.

AddVariable メソッドの引数仕様は下記のようになっています.

```
CaoController::AddVariable
(
    BSTR bstrName,                  // 変数名
    BSTR bstrOption                 // オプション
);
```

ここで, <変数名>は下記の文法で記述します.

<FOM のオブジェクト> [# <データ型>]

ここで<FOM のオブジェクト>は, InteractionClass のパラメータ名または, ObjectClass のアトリビュート名をフルパスで, “. ”区切りで指定します. FOM の仕様から, 最初の文字列は“InteractionRoot.”か“ObjectRoot.”のどちらかで始まります. また, ObjectClass の場合は“. ”区切りで指定したあとに“！”を付けてその後にインスタンス名(オブジェクト名)を指定します.

<データ型>はパラメータまたはアトリビュートのデータ型を数値で指定します.

表 2-3 に HLA プロバイダで指定できるデータ型一覧を示します. このデータ型を省略した場合は, 文字列(VT_BSTR)として扱われます. 例えば, " InteractionRoot.Communication.ABC#2" と指定すると, HLA プロバイダは ABC のデータを符号あり 2 バイト整数(VT_I2)として読み書きします. HLA の仕様ではデータのマーシャリングはフェデレートに任されているので, HLA プロバイダは CAO アプリケーションに代わってマーシャリングを行います. 従って, ここでデータ型を正確に指定し, かつ整数値の場合はオプション(4)でメモリ格納方式を合わせないと HLA プロバイダは RTI から正確に値を読み込んだり書き込んだりできないので注意が必要です.

表 2-3: 有効なデータ型

データ型	指定値	バイト数	説明
VT_I1		1	符号あり 1 バイト整数
VT_UI1	17	1	符号なし 1 バイト整数
VT_I2	2	2	符号あり 2 バイト整数
VT_UI2		2	符号なし 2 バイト整数
VT_I4	3	4	符号あり 4 バイト整数
VT_UI4		4	符号なし 4 バイト整数
VT_R4	4	4	単精度(32 ﾋﾞｯﾄ)浮動小数
VT_R8	5	8	倍精度(64 ﾋﾞｯﾄ)浮動小数
VT_BSTR	8	可変	文字列 VT_UI4 で示された文字数の後に UNICODE 文字と NULL ターミネータを付加
VT_BOOL	11	1	TRUE (-1), FALSE (0)

<オプション>は下記の文法で指定し, HLA プロバイダの動作を指定することができます.

Option=<オプション値>

ここで, <オプション値>は整数値で指定します. 表 2-4 に指定できるオプション一覧を示します. これらのオプションの和を取って複数指定することもできます.

表 2-4: 変数クラスのオプション

値	オプション名	対象	説明
1	イベントの無効化	Interaction	デフォルトでは Interaction メッセージはコントローラクラスの OnMessage イベントで通知されますが, このオプションを指定するとその代わりにキューに格納され, イベントは発生しません. 格納された値は, Value プロパティで取得できます.
2	オブジェクトの生成	Object	変数オブジェクトの生成時に指定された HLA のオブジェクトを作成します. このオプションが無効でかつ指定されたオブジェクトがその時点で存在しないと変数オブジェクトの生成に失敗します.
4	Big endian の変換	Interaction, Object	データの Read/Write で Big Endian に変換します. メモリ格納形式の違う通信相手と数値データを交換する場合に指定します.

8	所有権の解放禁止	Object	デフォルトでは他のフェデレートからオブジェクトの所有権を要求されたときに受け入れますが, このオプションを指定すると拒否します.
16	所有者なし所有権のプル	Object	デフォルトでは変数オブジェクトの値の書き込みが発生したときに所有権を侵略的に取得(Intrusive Pull)しようとします. このオプションを指定すると, だれも所有していない場合にのみ取得できます.
32	Publish 宣言なし	Interaction, Object	デフォルトでは変数オブジェクトの生成時に publish と subscribe の両方を宣言しますが, このオプションが指定されると publish を宣言しません. 但し, オブジェクトの生成オプション(2)がセットされている場合は, このオプションは無視されます.
64	Subscribe 宣言なし	Interaction, Object	デフォルトでは変数オブジェクトの生成時に publish と subscribe の両方を宣言しますが, このオプションが指定されると subscribe を宣言しません.

以下に AddVariable メソッドを実行するときの例を示します.
InteractionClass の場合は以下のとおりです.

```
AddVariable
(
    " InteractionRoot.Communication.Message ",
    " Option=32"                              // publish 宣言しません
);
```

ObjectClass の場合は以下のとおりです.

```
AddVariable
(
    " ObjectRoot.myClass.myAttr!XYZ "         // XYZ はオブジェクト名
    " Option=2 "                              // オブジェクトを新規に生成します
);
```

2.4.3. CaoController::Execute メソッド

Execute メソッドで有効なコマンドを表 2-5 に示します.

表 2-5: Execute コマンド

コマンド	引数	説明
startTimer	なし	イベント処理用スレッドの有効化【解説1】
stopTimer	なし	イベント処理用スレッドの無効化

【解説1】

HLA プロバイダ内部では HLA のインタラクションや同期ポイントのイベント処理用スレッドが動作しています. このスレッドは AddController メソッドを実行した時点で有効になります. 通常このスレッドの処理は AddVariable メソッド等でそのアプリケーションで必要なインタラクションやアトリビュートを追加した後で意味のあるものですので, それを一時停止したい場合には”stopTimer”コマンドを使います. 逆に開始したい場合には, ”startTimer”コマンドを使います.

2.4.4. CaoVariable::get_Value プロパティ

CAO 変数オブジェクトが HLA の ObjectClass である場合, get_Value で値をその現在値を取得することができます. また, InteractionClass で且つ CAO 変数クラスのオプションでイベントの無効化(1)を指定した場合は, キューに溜まっている Interaction メッセージを取得します.

2.4.5. CaoVariable::put_Value プロパティ

CAO 変数オブジェクトが HLA の ObjectClass である場合, put_Value でそのオブジェクトの対応する属性の現在値を設定することができます. また, InteractionClass である場合は Interaction メッセージを送信します. その場合 Interaction メッセージの tag には変数オブジェクトのコレクションをもつコントローラオブジェクトの ID を埋め込みます. これにより, そのメッセージを受信した方はそれがどのコントローラオブジェクトからのものなのかを識別することができます.

2.4.6. CaoVariable::get_ID プロパティ

CAO 変数オブジェクトが HLA の InteractionClass である場合, この ID プロパティには HLA プロバイダが自動的に与えた識別番号が格納されています.

2.4.7. CaoVariable::get_Help プロパティ

CAO 変数オブジェクトが HLA の InteractionClass である場合, この Help プロパティには受信した Interaction メッセージの tag 情報が格納されています. 送信したアプリケーションが CAO アプリケーションの場合は, 送信先の CAO コントローラオブジェクト[4]の識別 ID が格納されています.

[4] Ver.1.1.2 までは CAO 変数オブジェクトの識別 ID が格納されていました.

2.5. 変数一覧

2.5.1. コントローラクラス

前節で解説したように，HLA の FOM に定義されている Interaction や Object は，CAO の変数クラスにマッピングされています．CAO の変数クラスにはコントローラクラス，ロボットクラスなど幾つかありますが，HLA プロバイダはその中でコントローラクラスの変数を使用します．

CAO の変数には，ユーザ変数とシステム変数があります．HLA プロバイダはコントローラクラスのユーザ変数に FOM に定義されている Interaction や Object を割り当て，システム変数に HLA の動作を制御するようなシステム変数を割り当てています．それぞれの詳細は下表で説明します．

表 2-6：コントローラクラス ユーザ変数一覧

変数名	データ型	説明	属性	
			get	put
<任意の Interaction>	<表 2-3 の型>	FOM に定義されている任意の Interaction を変数として扱います．	○	○
<任意の Object>	<表 2-3 の型>	FOM に定義されている任意の Interaction を変数として扱います．	○	○

表 2-7：コントローラクラス システム変数一覧

変数名	データ型	説明	属性	
			get	put
@CURRENT_TIME	VT_R8	フェデレートの論理現在時刻．値を書き込むと現在時刻が修正されます．	○	○
@ADVANCE_TIME	VT_R8	フェデレートの論理現在時刻を指定時間だけ進めます．	-	○
@LOOKAHEAD	VT_R8	フェデレートの LookAhead 時間．	○	○
@LBTS	VT_R8	LBTS (Lower Bound Time Stamp) 時刻．	○	-
@NEXTEVENT_TIME	VT_R8	先頭の TSO イベント時刻 (Minimum Next Event Time)．	○	-
@SYNC_POINT	VT_BSTR	フェデレーションの同期ポイントの設定．この値を設定すると，OnMessage(1)イベントが発生します．	-	○

2.6. イベント

2.6.1. CaoController::OnMessage イベント

冒頭の設計方針で解説したように，HLA プロバイダは HLA のすべてイベントを単純にラップするような設計ではなく，出来る限り自動的に管理し，クライアントが必要な最低限のイベントだけ CaoMessage オブジェクトでカプセル化して OnMessage イベントで渡します．イベントの種類は CaoMessage オブジェクトの Number プロパティで判別できます（表 2-8）．

表 2-8： OnMessage イベントのメッセージ一覧

番号	メッセージ			応答	
Number	Value	Destination	Source	Clear	Reply
1	同期メッセージ[5]	<Empty>	<Empty>	同期完了の通知	-
2	<Empty>[6]	<Empty>	<Empty>	-	-
3	Interaction メッセージ[7]	受信した変数オブジェクトの ID	送信元コントローラオブジェクトの ID	-	Interaction メッセージの送信[8]

2.7. エラーコード

CRD プロバイダでは，固有のエラーコードはありません．ORiN2 共通エラーについては，「ORiN2 プログラミングガイド」のエラーコードの章を参照してください．

[5] HLA から同期ポイントの通知（announceSynchronizationPoint）があった場合にこのイベントが発生します．
[6] HLA から同期完了の通知（federationSynchronized）があった場合にこのイベントが発生します．
[7] HLA から Interaction メッセージを受信（receiveInteraction）した場合にこのイベントが発生します．ただし，AddVariable のオプションでイベントの無効化(1)がセットされている場合は，このイベントは発生しません．
[8] HLA の Interaction メッセージのコンテキストを保存して返信することができます．

3. サンプルプログラム

以下に簡単な Visual Basic のサンプルプログラムと実行結果を示します.

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
```

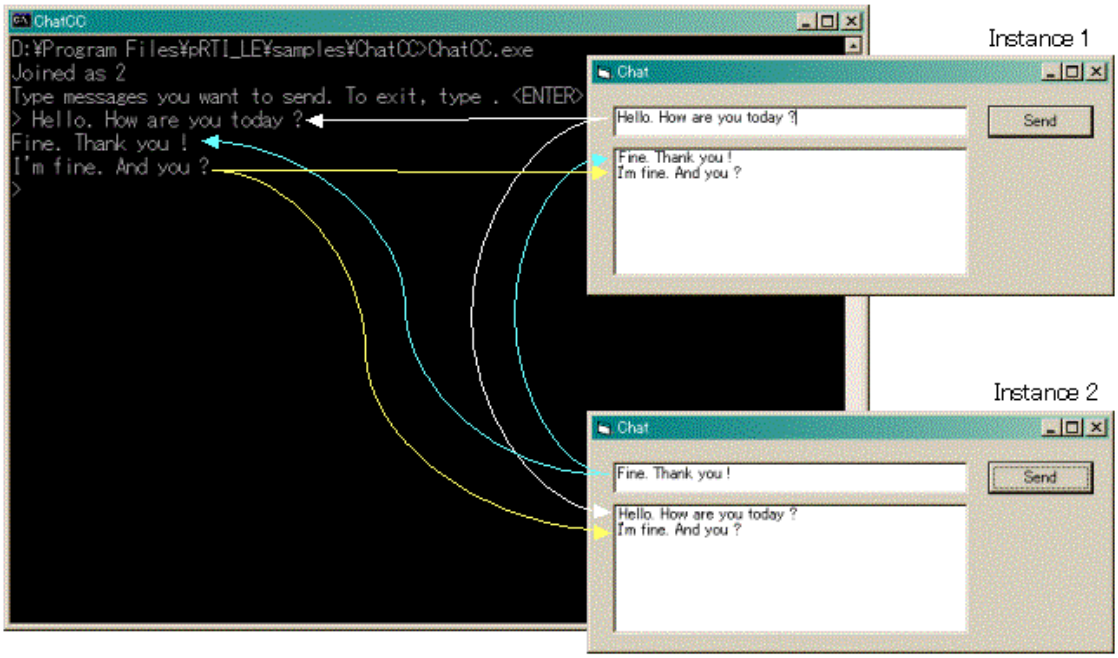

図 3-1：サンプルの実行結果

このサンプルプログラムは Pitch AB (http://www.pitch.se/) の pRTI Limited Edition に付属のサンプルプログラム(ChatCC)と同様な動きをします．二つを比較すると，HLA プロバイダのメカニズムをより深く理解できます．図 3-1 に示したように，このプログラム同士はもちろん，付属のサンプルプログラムとも HLA のメッセージを簡単に交換することができます．

【参考】

上記のサンプルプログラムでは HLA プロバイダのすべての機能はわかりませんので，別途テストプログラムを容易しています(図 3-2)．このプログラムのソースコードを参照すれば HLA プロバイダの使い方をより深く理解できます．

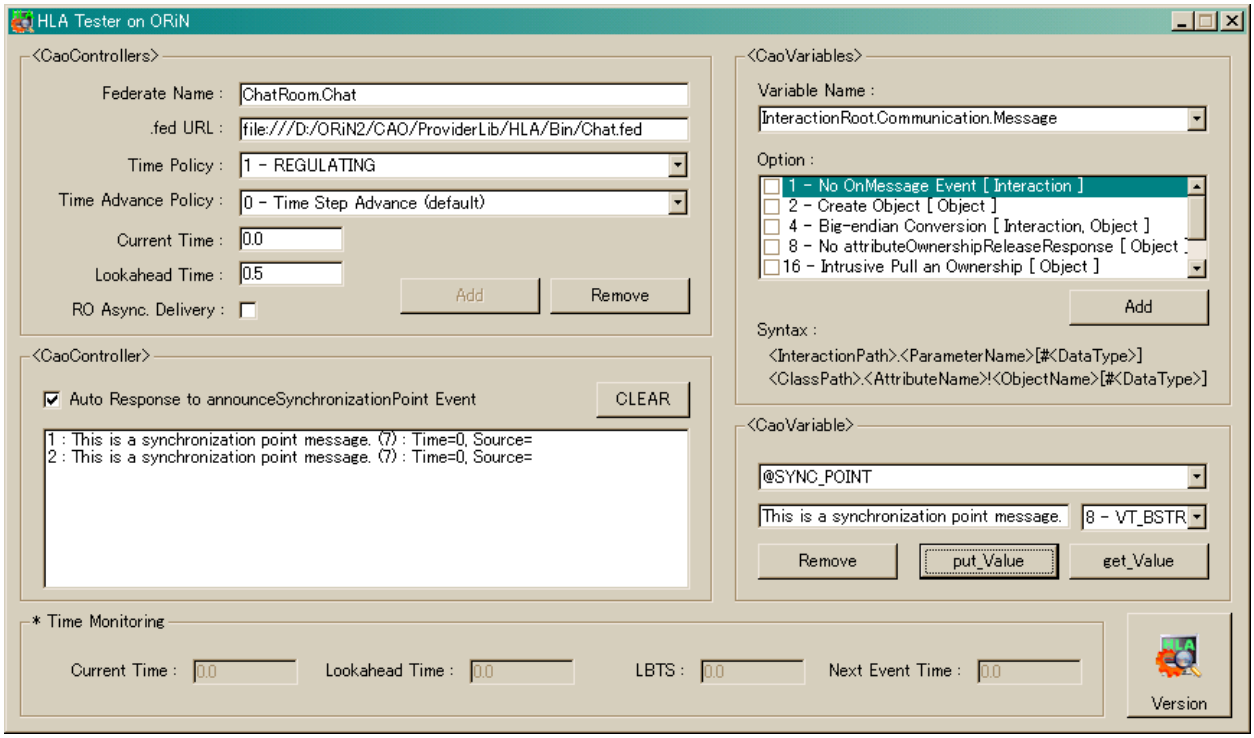

図 3-2：テストプログラム

4. 内部動作メカニズム

これまでの章では HLA プロバイダのクライアント(CAO アプリケーション)からの使い方に焦点をあてて説明してきました. 本章では, その理解を更に深めるために, HLA プロバイダ内部で HLA の機能がどのように使われているかを主なものを簡単に解説します.

HLA の機能を大別すると, 下記のようになります.

- フェデレーション管理 (Federation Management)
- デクラレーション管理 (Declaration Management)
- オブジェクト管理 (Object Management)
- 所有権管理 (Ownership Management)
- 時刻管理 (Time Management)
- データ分散管理 (Data Distribution Management)

ここで, データ分散管理の機能は単に通信量を減らすためのメカニズムなので ver.1.0 ではこの機能は使いません. その他の機能がどのタイミングで使われるかを順に簡単に解説します.

4.1. フェデレーション管理

この機能はフェデレーション全体の実行管理を行う機能で, 各フェデレートの管理や, メッセージのルーティング, 同期管理などを行います. これらの機能は, CAO のコントローラオブジェクト(CaoController)にマッピングされています. つまり, CaoController オブジェクトが HLA のフェデレートに対応します.

具体的には, CaoWorkspace クラスの AddController メソッドを呼び出したときに, HLA プロバイダ内部でフェデレーションを作成, または既に作成されている場合はそのフェデレーションに参加します. 逆に CaoController オブジェクトが消滅するときに, そのフェデレーションから抜けます(図 4-1). このタイミングで HLA の管理機能のクリアされますので, 消滅した後に HLA と通信することはできません.

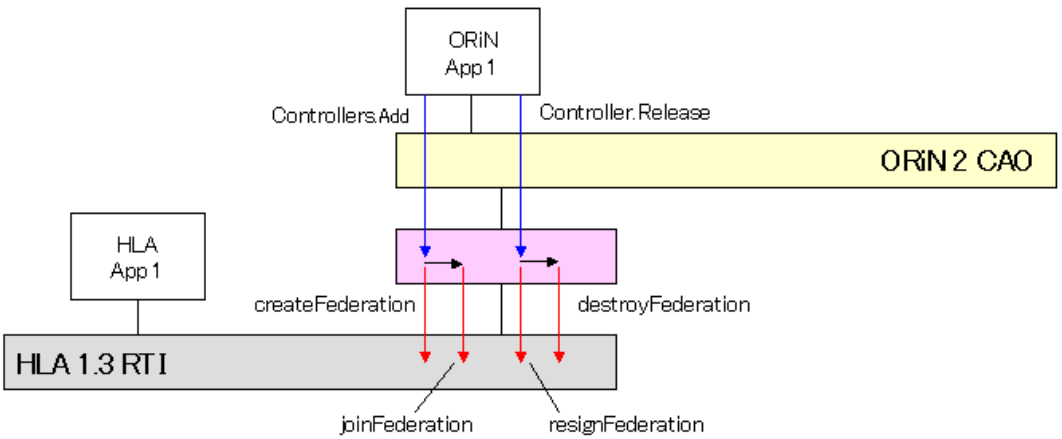

図 4-1: フェデレートのライフサイクル

HLA では, 各フェデレートの同期を簡単に取るための機能(Synchronization Point)が実装されています.

これを使うことで, CAO アプリケーションと, HLA アプリケーションで同じく同期を取ることができます(図 4-2). もちろん, この機能を使って CAO アプリケーション間で同期を取ることも可能です.

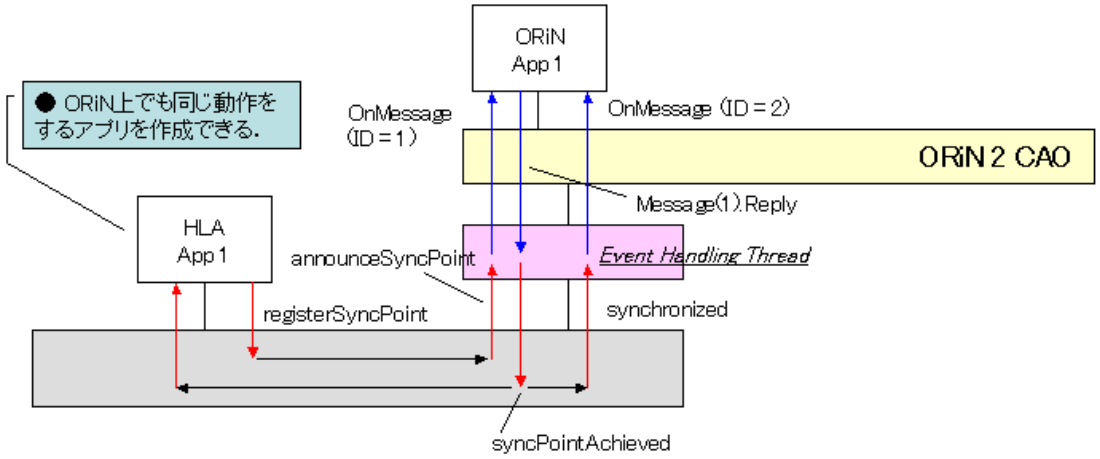

図 4-2: フェデレートの同期

4.2. デクラレーション管理

この機能はインタラクション(Interaction)やクラスオブジェクト(Object)の各種宣言を管理します. 例えば, publish と subscribe の宣言などです. HLA のインタラクションやオブジェクトを使うとき, 各フェデレートはそれを生成し値を更新する(publish)のか, 参照するだけ(subscribe)なのかを正確に宣言しなくてはなりません.

この機能は, いわゆるオブジェクト指向の世界にもないアイデアだと思われます. HLA の資料によるとこれも通信量を減らすための一つのアイデアであるとのことですが, クライアントプログラムからするとそれだけ手続きが増えてしまいます. 従って, HLA プロバイダではこの宣言も内部で自動的に行います.

具体的には, CaoController クラスの AddVariable メソッドを呼び出したときに, つまり変数オブジェクトを生成するときに宣言します. デフォルトでは, publish と subscribe の両方を宣言します(図 4-3, 図 4-4). もし, 明らかに変数オブジェクトの Value プロパティに対して書き込みしかしない, 逆に読み込みしかしない場合は, AddVariable のオプションで指定することができます(表 2-4). これらの宣言によって, CAO の変数オブジェクトの属性値(Attribute プロパティ)は表 4-1 のようになります.

HLA のオブジェクトはそのアトリビュートごとにこの宣言をすることができます. つまり, もし既に生成されているオブジェクトの別のアトリビュートであった場合は, AHS(Attribute Handle Set)を作成しなおして再度宣言しなおさなくてはなりません. HLA プロバイダは, 一つのアトリビュートに一つの変数オブジェクトを対応させますので, 常にこのようなことが発生する可能性があります. この動的な再宣言も HLA プロバイダ内部で自動的に行われます.

表 4-1: CAO 変数オブジェクトの属性値

HLA の宣言	変数の属性値
Subscribe	1 – 読み込み可
Publish	2 – 書き込み可
Subscribe & Publish	3 – 読み書き可

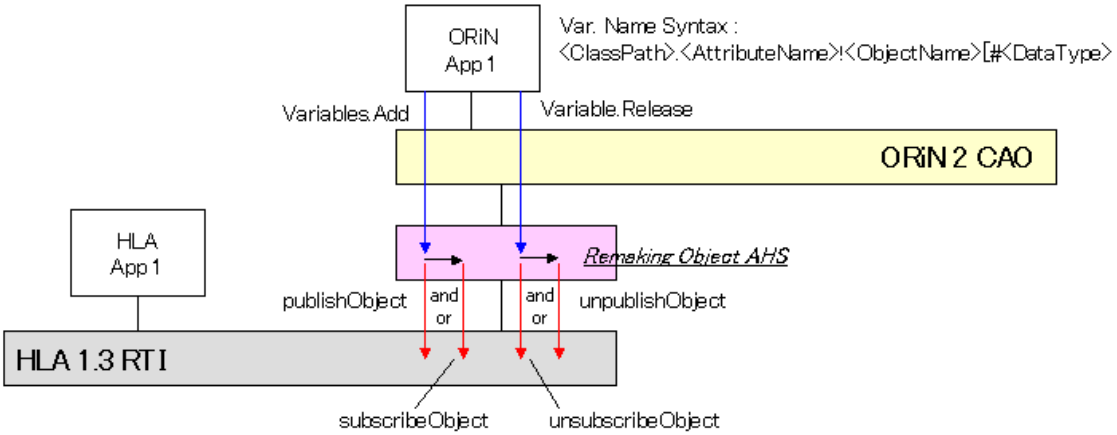

図 4-3：オブジェクトの宣言

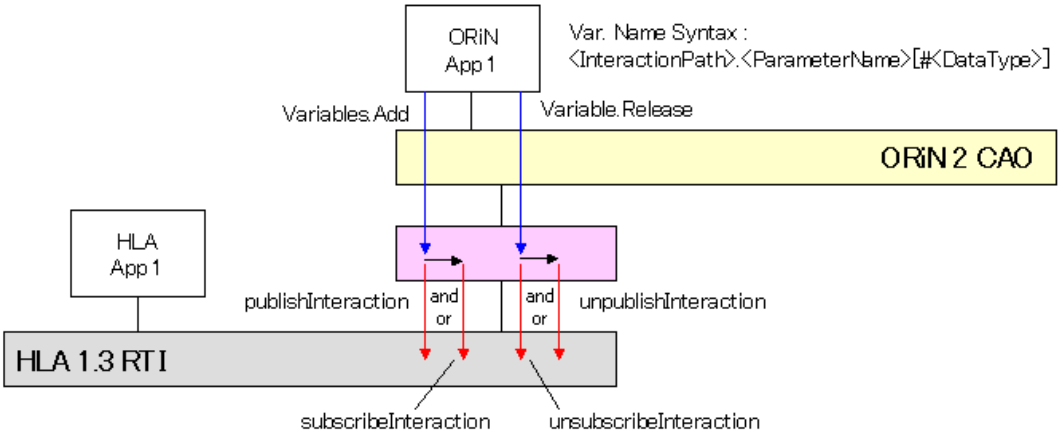

図 4-4：インタラクションの宣言

4.3. オブジェクト管理

この機能はインタラクションやクラスオブジェクトのライフサイクルや更新・参照などの管理を行います. 前述したように, HLA のインタラクションやクラスオブジェクトは CAO の変数オブジェクトにマッピングされていますので, これらの機能は変数オブジェクトの生成・削除, 値の読み書きに対応します(図 4-5).

HLA オブジェクトのアトリビュートの値は HLA プロバイダの中でキャッシュされますので, CAO アプリケーションは任意のタイミングで値を参照できます.

また, HLA のデータ交換の仕組みは単純にメモリイメージを転送しているらしく, そのマーシャリング処理はフェデレートが行わなくてなりません. HLA プロバイダはそのデータ型やメモリ格納方式に応じてマーシャリング処理も自動的に行いますが, 正確にマーシャリングするために AddVariable メソッドの実行時にデータ型や格納方式を指定しなくてはなりません. データ型は変数名で, 格納方式はオプションで指定します.

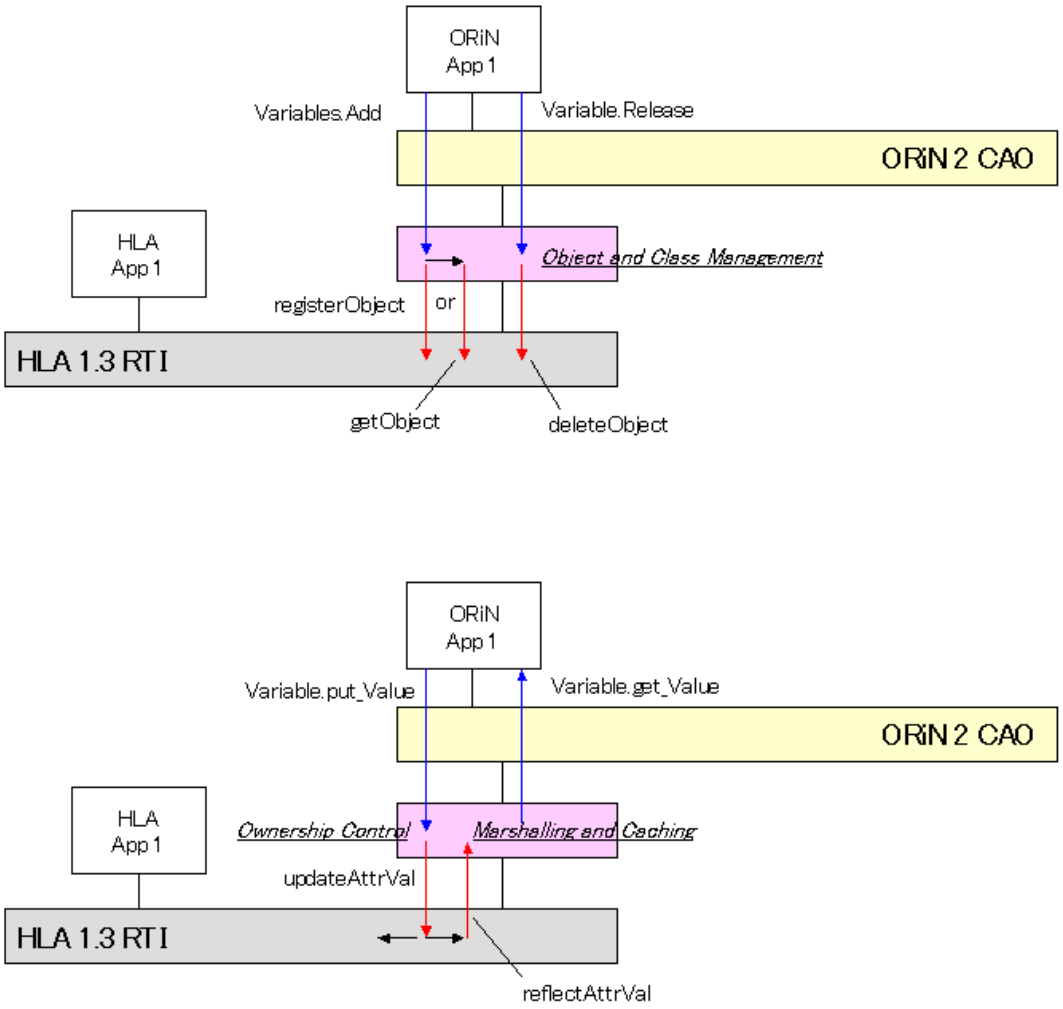

図 4-5: オブジェクトアトリビュートの更新と参照

HAL インタラクションはデフォルトではキャッシュ(インタラクションの場合は正確にはキューイング)されません. 受信した時点で OnMessage イベントを発行します(図 4-6). キューイングするように動作を変更する場合

は，AddVariable のオプションでイベントの無効化(1)をセットします．

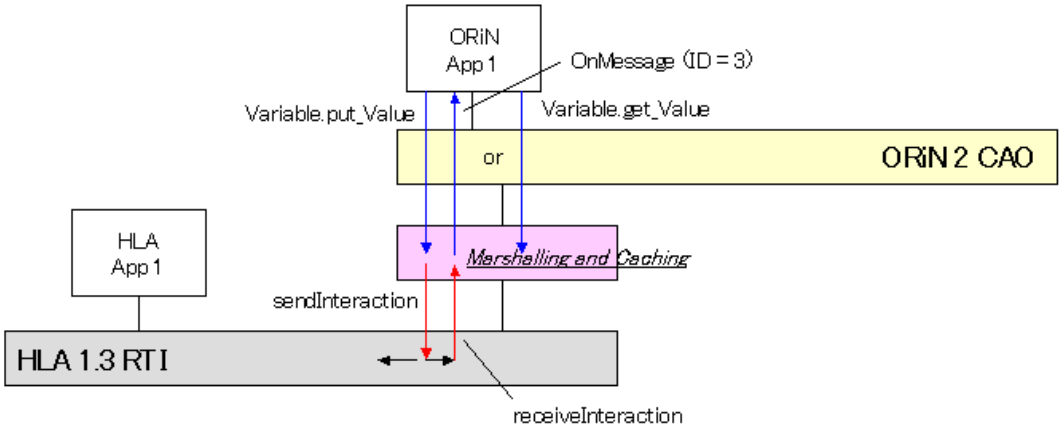

図 4-6：インタラクションパラメータの送信と受信

CAO 変数(CaoVariable)オブジェクトにマップされた HLA インタラクションは，メッセージを発行する際に自動的にメッセージの tag にその変数オブジェクトのコレクションを所有するコントローラオブジェクトの識別 ID が付加されます．そして，このメッセージを格納した CAO メッセージ(CaoMessage)オブジェクトの Source プロパティにはその ID が格納されています（図 4-7）．この仕組みにより“誰が”発行したメッセージであるかを受け側が判別できるようになるので一つのインタラクションを複数のフェデレートで共有する場合に役立ちます．

また，CAO メッセージオブジェクトの Reply メソッドを利用するとメッセージを受信した CAO 変数オブジェクトの put_Value が呼び出され，同様にその Reply メッセージを各フェデレートは受信することができます．

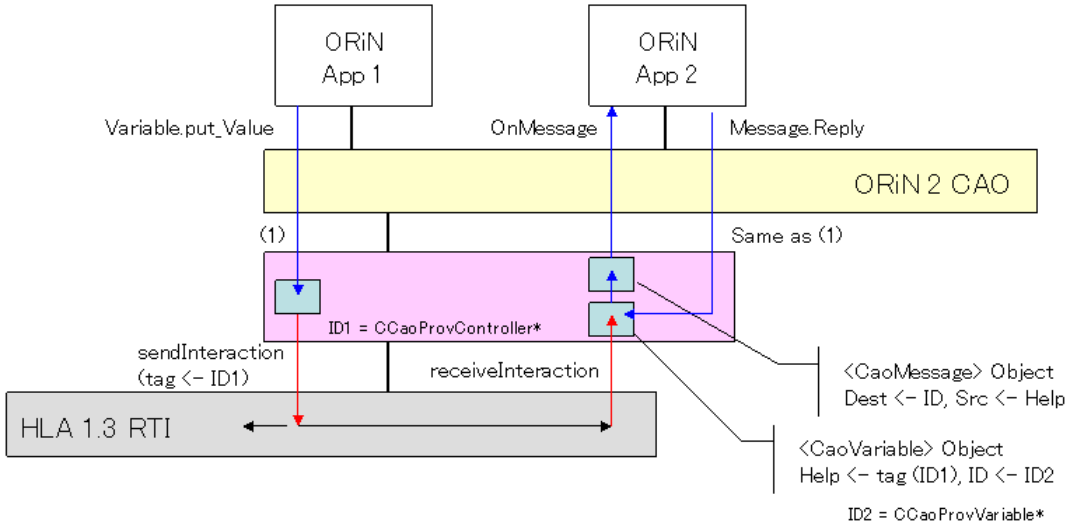

図 4-7：インタラクションの共有

4.4. 所有権管理

この機能はクラスオブジェクトの所有権の管理を行います. この所有権はオブジェクトのアトリビュートごとに管理されており, 所有権がないとアトリビュート値の更新などができません. つまり, どの時刻においても唯一のフェデレートしか値の更新ができないのです. この所有権はフェデレート間でネゴシエーションしてその権限を取得したり, 逆に自分が持っている所有権を解放したりできます.

これは, そもそもいわゆるオブジェクト指向の世界にもない概念だと思われます. HLA の資料によるとこれは通信量を減らすための一つのアイデアであるとのことですが, クライアントプログラムからするとそれだけ手続きが増えてしまいます. HLA プロバイダはこのような所有権の管理をプロバイダの内部で自動的に行うことによって, そのクライアントに所有権を意識させません.

具体的には, 変数値の書き込みのタイミングで自分が所有権を持っていないときは, HLA プロバイダは所有権を取得してから書き込みます. その取得のポリシーはデフォルトでは侵略的プル(Intrusive Pull)です. AddVariable のオプション(16)をセットした場合は, だれも所有していない場合にのみ取得できます. 取得できなかった場合はエラーを返します.

また, 逆に他のフェデレートから所有権を要求された場合はそれに応答して解放します. オプション(8)をセットするとこの解放が禁止されます. その場合は, 所有権を取得しようとはしますが, 自分は一切解放の要求にこたえないので, 時刻が進むにつれて, 所有権が HLA プロバイダで作成したフェデレートに集中することになります.

4.5. 時刻管理

この機能が HLA の最も特長的な機能で，各フェデレートの論理時刻の管理を行います．すべてのフェデレートは独立に論理時刻を持つことができます．フェデレーション全体のユニバーサルタイムはありません．

この時刻管理には図 4-8 に示す論理時刻に関する様々な項目があります．それらは，CAO の変数オブジェクト(CaoVariable)にマッピングされています．この変数オブジェクトの値を設定することで，論理時刻を進めたりします．

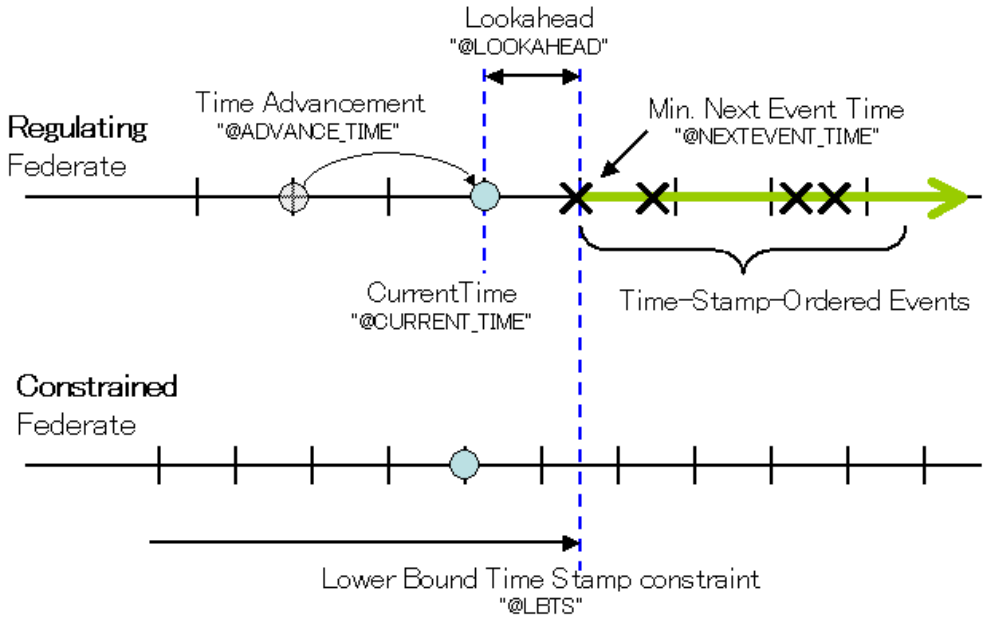

図 4-8：時刻管理

すべてのフェデレートは次の 4 つのいずれかの時刻管理ポリシーを持ちます．

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

この中で，1 は時刻に関して何の制約も受けません．“REGULATING”フェデレートのみが，TSO(Time Stamp Order)イベントを発行することができます．具体的には，論理時刻付きのインタラクションパラメータの送信や，オブジェクトアトリビュートの更新などができます．“CONSTRAINED”フェデレートはその TSO イベントを取得することができます．

CAO アプリケーションはこの 4 つのタイプのフェデレートを作成することができます．具体的には，CaoController オブジェクトを生成するときに，AddController のオプション(TimePoli)で指定します(図 4-9)．時刻を進めるには，システム変数の“@CURRENT_TIME”や“@ADVANCE_TIME”の値を設定することで進めることができます．

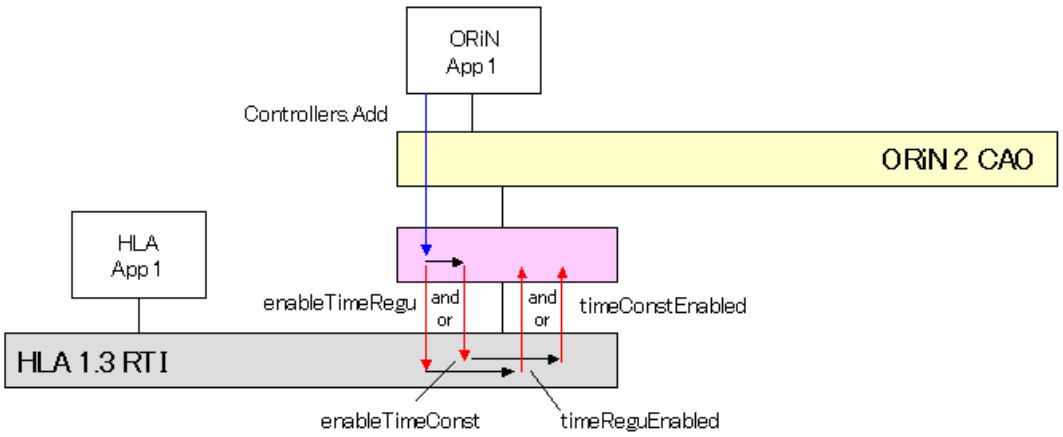

図 4-9：時刻管理ポリシー

HLA の時刻の進め方には，下記の 3 通りがあります．それぞれの進め方に応じて，取得できる TSO イベントが違います．

(1) Time Step Advance (TAR)

(2) Event-based Advance (NER)

(3) Optimistic Advance (flushQueue)

この論理時刻の進め方も AddController のオプション(TimeAdv)で指定します．このように，時刻管理ポリシーや論理時刻の進め方は CaoController オブジェクトを生成する時点で決められ，後でそれを変更することはできません．その場合には CaoController オブジェクトを生成しなおす必要があります．

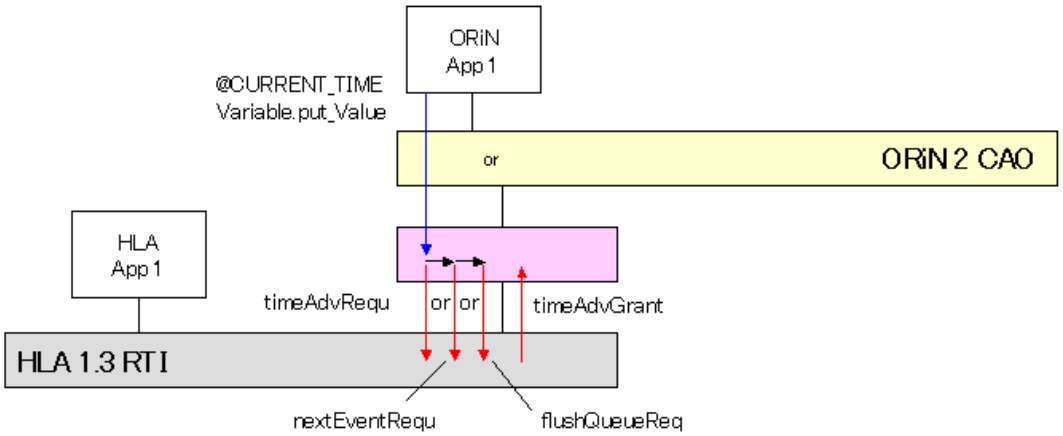

図 4-10：論理時刻の進め方