ISSN 1349-1113
JAXA-RR-04-017

宇宙航空研究開発機構研究開発報告
JAXA Research and Development Report

宇宙機シミュレーションのための分散オブジェクト指向フレームワークの研究

高橋　孝,　上田　裕子,　平野　聡
邑中　雅樹,　Qu Runtao,　小堀　壮彦

2005年1月

宇宙航空研究開発機構
Japan Aerospace Exploration Agency

JAXA-RR-04-017

宇宙航空研究開発機構研究開発報告
JAXA Research and Development Report

宇宙機シミュレーションのための分散オブジェクト指向フレームワークの研究

Study on Distributed Object-Oriented Framework for Spacecraft Simulation

高橋　孝[*1]、上田　裕子[*1]、平野　聡[*2]、邑中　雅樹[*2]、Qu Runtao[*2]、小堀　壮彦[*3]
Takashi TAKAHASHI , Hiroko UEDA , Satoshi HIRANO , Masaki MURANAKA , Qu Runtao , Takehiko KOBORI

*1: 総合技術研究本部 情報技術開発共同センター
Information Technology Center
Institute of Space Technology and Aeronautics

*2: 産業技術総合研究所
Advanced Industrial Science and Technology

*3: 三菱スペース・ソフトウエア株式会社
Mitsubishi Space Software Co.,Ltd.

2005年1月
January 2005

宇宙航空研究開発機構
Japan Aerospace Exploration Agency

宇宙機シミュレーションのための
分散オブジェクト指向フレームワークの研究

高橋　孝, 上田裕子(宇宙航空研究開発機構),
平野　聡, 邑中雅樹, Qu Runtao(産業技術総合研究所),
小堀壮彦(三菱スペース・ソフトウエア)

Study on Distributed Object-Oriented Framework for Spacecraft Simulation

Takashi Takahashi, Hiroko O. Ueda (JAXA),
Satoshi Hirano, Masaki Muranaka, Qu Runtao (AIST),
and Takehiko Kobori (MSS)

Abstract

We aim that a simulation framework of Spacecraft Simulation Environment (SSE) can be commonly applied not only to Full Software Simulations (FSS) but also to Processor-In-the-Loop Simulations (PILS) and to Hardware-In-the-Loop Simulations (HILS), while various spacecraft simulators are generally tailor-made at individual phases of development. Prior to the actual implementation of SSE, we designed the framework for FSS and PILS, and demonstrated its advantage to spacecraft simulations using an experimental system.

In this study, we implemented an experimental system for PILS, and demonstrated feasibility of the framework using the system. Real-time tasks working on μITRON communicate with the rest of spacecraft simulator through Java-based middleware "Hirano's Object Request Broker (HORB)"[1] via distributed communication interfaces (I/Fs) written in Java. These I/Fs are designed to be commonly applied to both FSS and PILS. We also implemented a tool called "Java-μITRON Bridge GENerator (JBGEN)" to automatically generate Java-μITRON communication programs from the I/Fs.

Furthermore, "MemorySaving" HORB is developed in order to avoid communication latency.

1.　はじめに

人工衛星など宇宙機の開発においてシミュレータは，図 1-1に示すようにシステム，サブシステム，機器コンポーネントの設計開発，運用手順確認，運用要員訓練などに利用される．宇宙航空研究開発機構(以下，「JAXA」)では従来，これらはそれぞれ用途ごとに開発されてきた．そこでJAXA 情報技術開発共同センター(以下，「IT センター」)ではこれらとは異なり，宇宙機の開発ライフサイクルを通じて，また複数の宇宙機に共通して利用される汎用的なシミュレーション支援環境の研究を行っている．またシミュレーション環境の要素技術として，複数計算機に分散したモジュー

ルが連係してシミュレーションを行うための仕組みについての研究も行っており，これまでに分散通信ミドルウェアの一つであるHLA/RTI(High-Level Architecture/Run-Time Infrastructure)の宇宙機シミュレーションへの適応性評価を目的として，ベンチマーク試験および簡易宇宙機シミュレータを用いたフル・ソフトウェア・シミュレーション(FSS: Full Software Simulation)を行ってきた[2].

一方，宇宙機のシミュレーション形態は，表 1-1に示すように FSS だけでなく，衛星の姿勢軌道制御系(AOCS: Attitude and Orbit Control Subsystem)やデータ・ハンドリング系(DH: Data Handling subsystem)などリアルタイムで動作する搭載計算機(OBC: OnBoard Computer)および搭載ソフトウェア(OBS: OnBoard Software)を分散オブジェクトとするシミュレーション(PILS: Processor-In-the-Loop Simulation)や，実際の運用卓，搭載ハードウェアと連係するシミュレーション(HILS: Hardware-In-the-Loop Simulation)がある．このような様々なシミュレーション形態を共通の枠組みで実現する技術の確立を目標として，平成14年度後期から平成15年度の間，組込みシステムや分散オブジェクト技術を専門とする産業技術総合研究所(以下，「産総研」)情報処理研究部門の平野博士を中心とするグループとの共同研究を行った．

FSS, PILS, HILS などの異なるシミュレーション形態に対応したシミュレーション環境に関する研究は海外の宇宙機関でもなされており，以下のような事例がある．

- ESA: 宇宙機シミュレーションの実行環境であるEuroSim[3], ESAが開発したプロセッサのソフトウェア・エミュレータを備えたOBS検証環境であるSVF(Software Validation Facility)[4], および HILS を行う TVE(Test and Verification Equipment)[4]等があり，それらを統合するVS-RF(Virtual Satellite Reference Facility)[5]の構想がある．
- EADS Astrium: モデル・ベース開発により宇宙機ライフサイクルを通じて一貫して利用可能な開発・試験環境であるMDVE(Model-based Development & Verification Environment)[6]が実働しており，発展を続けている．

また，宇宙機のOBSをJavaで開発するプロジェクトが以下のように進められている．

- ESA: 衛星メーカの Astrium，ソフトウェア・メーカの aicas と共同してリアルタイム仕様のJavaVM(RTSJ: Real-Time Specification for Java)の開発[7]を行い，Proba2プロジェクトで応用される計画がある．
- JPL: Sun, Carnegie Mellon 大学，TimeSysと共同でRTSJのJavaVMを開発しており，Golden Gate Projectと呼ばれるプロジェクト[8]においてOBS開発の検討が行われている．

本研究では，FSSだけでなくPILSに関しても共通的に利用できるシミュレーション・フレームワークを構築するために，平野らが開発した Java 分散ミドルウェア HORB(Hirano's Object Request Broker)[1]を宇宙機シミュレータに適用して，リアルタイム OS(RTOS: Real-Time Operating System)上で動作するOBSとWindows™やLinux™などの汎用OS上で動作する宇宙機シミュレータが連係する分散シミュレーションの実現性，実用性の評価検討を行った．これには既存の簡易HTV(H-IIA Transfer Vehicle)/ISS(International Space Station)シミュレータのモジュールを利用してFSSとPILSが共通インターフェイス(I/F)で実現可能なフレームワークを設計，Java言語で実装，および整備した実験装置を利用して動作確認を行った．またOBSとシミュレータ間の通信I/Fの作成を容易にするため，通信 I/F プログラムの自動生成ツール JBGEN(Java-μITRON Bridge

GENerator)を開発した．また，HORB 自身のリアルタイム拡張，多言語対応も行われた．

さらに，次世代シミュレーション実行制御の基礎となる論理時間スケジューラの設計および部分試作を行った．本スケジューラは，並行プログラミング技術の一つであるマルチスレッド技術を利用して設計され，論理時間でスレッドが切り替わる機能の試作と動作確認を行った．

以下，2 章では分散通信ミドルウェア HORB について述べ，3 章には検討を行った OBS と宇宙機シミュレーション環境との通信フレームワークについて記述する。また 4 章には論理時間スケジューラについて示し，最後にまとめと今後の課題について述べる。

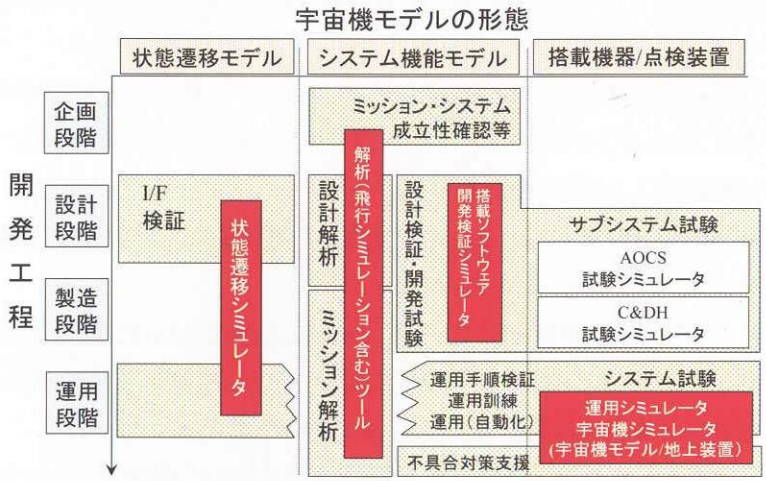


図 1-1 宇宙機開発におけるシミュレータ利用

表 1-1 宇宙機シミュレーションの形態と分散オブジェクトの候補

シミュレーション形態	分散オブジェクト	目的	分散ハードウェア
FSS (Full Software Simulation)	他の宇宙機シミュレータ 運用卓モデル、地上局モデル ミッション機器モデル AOCSモデル、C&DHモデル 解析エンジン・モデル	ランデブ飛行解析 運用模擬 ミッション検証 搭載ロジック検証 負荷分散	計算機
PILS (Processor in the Loop Simulation)	AOCS、C&DH ミッション系制御・C&DH機能	搭載ソフトウェア検証	搭載CPU評価ボード 搭載CPUボード
HILS (Hardware in the Loop Simulation)	運用卓 AOCS、C&DH、ミッション系等コンポーネント センサ アクチュエータ	運用性検証 搭載機器検証	運用卓 搭載コンポーネント センサ アクチュエータ

2. 分散通信ミドルウェア

宇宙機シミュレータは今後，宇宙機のライフサイクルの多岐に渡る様々な処理を扱うようになると予想される．シミュレータの利用者，即ちシミュレーション対象の宇宙機の特性やライフサイクル中の処理内容を記述する人は必ずしもプログラミングのスキルに長けているとは限らない．そのため，標準的な開発者が開発に参加していても，生産性と信頼性の高いシステムを構築することが可能な枠組みを作ってゆく必要がある．Java や分散オブジェクトのような技術は性能上のコストがかかるが，生産性や信頼性をもたらす．CPU やメモリの性能向上と，低消費電力化が進んでおり，そのコストを払うことができるようになってきた．長期に渡って使用可能な枠組みを構築するためには，既存の技術と先進的な技術を両立しつつ，徐々により進んだソフトウェア技術を取り入れることができなければならない．

本章では，まず背景として本システムで採用したオブジェクト指向言語と C 言語の併用の理由について延べ，分散オブジェクトのようなオートコード(プログラムの自動生成)技術が必要である理由を説明する．その概要は以下の通りである．

- 生産性と信頼性を向上するには，Java に代表される現代的なプログラミング言語と組織的なシステム開発プロセスを取り入れる必要がある．
- しかし，既存のソフトウェア資産を利用したり，Java に欠けているリアルタイム性を満足するためには，C 言語とリアルタイム OS も用いなければならない．複数の OS や複数のプログラミング言語，また複数の計算機を併用することは，適材適所と言える．
- 一方で，どの機能をどの計算機上でどの言語で実現するかは開発のフェーズや時代によって変わるため，固定的な物ではない．言語間，OS 間，計算機間にまたがる機能を接続したり機能を移動することは，工数とエラーの増加をもたらすという課題が生ずる．
- そこで，その課題を克服するため，本研究では分散オブジェクトのような仕様からプログラムを自動生成するオートコード技術を可能な限り取り入れた．

本章の後半では，分散オブジェクト技術 HORB の原理，特徴，他の分散オブジェクトとの比較，リアルタイム性の拡張について述べる．

2.1 職人芸的手工業から組織的な方法論に移行している組込みソフトウェア開発

プログラミング言語の進化

一般に，システム開発において機能の要求はますます複雑化する一方で，開発期間やコストは最低限に抑えなければならない．そのような要件を満たすため，オブジェクト指向プログラミングが普及し，高度なユーザインターフェースを備える高機能なプログラムを開発可能なプログラム開発環境が利用可能になった．

オブジェクト指向言語としては長くC++言語が使用されてきたが，ポインタの予期せぬ不正なメモリ操作，メモリ管理の困難さやマルチスレッド(マルチタスク)機能の欠如といった難しさがあり，すべてのバグ(不具合)を克服して信頼性の高いプログラムを開発することはかなりの力量と長い開発期間を要する作業であった．また，ウィルスの温床になるなどセキュリティ面での問題が絶えない．そこで，C++ や伝統的な C 言語に代わり，Java や C#のように当初から自動メモリ管理やマルチス

レッド等の高信頼性のための言語機能を備えるオブジェクト指向言語が一般的になってきた．大学のプログラミング教育ではJavaが用いられることが多く，最近の技術者にとってはJavaが第一選択言語となりつつある．文法的にはどれもC言語のファミリーに属しているため，ベテランの技術者にとっても習得や移行は容易である．

モデリングとテストを伴う開発方法論の重要性

オブジェクト指向開発は現実の複雑な要求を分析し適切なプログラム構造を見いだすモデリングと，記述したプログラムコードの正しさをテストによって検証しながら開発を進める方法論(開発プロセス)を伴っており，従来の言語と比較して短期間で信頼性のあるプログラムの作成が可能である．金融分野や商取引などミッション・クリティカルな分野ではJavaの利用が広がっており，多くの実績がある．

組織としてのシステム開発能力は，開発プロセスがどれだけ組織に浸透し習熟が進んでいるかを示すCMM(能力習熟度モデル)レベルで表される．例えば国際宇宙ステーションのソフトウェア開発ではCMMの最高レベル5の達成を目指すとしている．我が国の政府調達でもCMMの実績を加味することが求められている[13,14]．

組込みソフトウェア開発技術の動向

本研究では，PILSとしてワークステーション上のシミュレータ本体と搭載CPUボード上のOBSを組み合わせてシミュレーションを行う．一般に，OBSが属する組込みソフトウェア開発の分野では，ソフトウェア工学や最近のプログラミング言語の技術や知見が十分に活かされていないと指摘されている[15]．

リアルタイムOSを用いるマルチタスクの組込みソフトウェア開発は難しいと言われ，品質は開発者個人の技量に大きく依存する．プログラムの構造を把握し検証をすることが難しく，デバッグも困難である．リアルタイムOSを用いずにCとアセンブラでステートマシンと割り込み処理を記述することは更に難しい．そのため，ソフトウェア工学に基づく現代的で組織的な開発方法論と言語への移行が進みつつある．

JavaとCの併用とオートコード

Javaの実行プログラムは中間コード形式であるためCPUやハードウェアには依存しない．ハードウェア構成が異なる様々な機種が存在してもJava処理系がハードウェアの違いを吸収するため，開発者は共通のJavaプログラムを使い続けることができる．このことは，本研究で目的とする複数の衛星に共通のシミュレーション基盤を確立する際に重要な利点となる．しかし，組込みソフトウェア開発においてはJavaにはいくつかの欠点がある．

- 自動メモリ管理(ガベージコレクション)が発生すると処理が中断されるため，リアルタイム性に欠ける．
- インタプリタ型のJavaVM(Java処理系)の場合，性能が劣る．
- ガベージコレクションの抑制や割り込み処理といったJavaのリアルタイム化は目下進展中であり，今日すぐに使用できるリアルタイムJava処理系は限られている．
- C/C++で記述された既存のソフトウェア資産を使いたい．例えば，制御系モデルの記述に使用したMATLAB™はCのプログラムコードを生成する．

そこで本研究では，汎用計算機上で実行されるシミュレータ本体はJavaで記述し，必要に応じてJavaからC/C++で記述された既存のアプリケーションを呼び出す形を用いた．搭載模擬装置ではリアルタイムOSとしてμITRONを搭載し，ITRON上でJava処理系及びCのリアルタイムタスクを両方動作させることでプログラミング性とリアルタイム性の両立を図ることを試みた．

現状では搭載模擬装置上のJava処理系のリアルタイム性が低いことが問題となり次章で述べるような対策をとったが，中長期的にはJavaのリアルタイム性能は向上していくため，JavaとCの両方を混在させる構成は開発者に柔軟性を提供し，開発期間や信頼性のトレードオフを勘案しながら徐々にJavaに移行することを可能とする．

複数の言語やOSを用いることにより信頼性が損なわれたり開発期間が長くなることがあっては本末転倒である．開発者はシミュレーション・モデルのロジックに集中したい．そこで次章で述べるように，本研究では仕様からプログラムを自動生成するオートコードを可能な限り取り入れた．シミュレータ全体はフレームワーク構築とロジックのコンポーネント化が容易なJavaで記述し，制御系モデルはMATLAB™からCのプログラムコードを自動生成する．同一プロセス内のJavaからCの接続のプログラムコードはJNIを用いて生成した．μITRON上のJavaスレッドからCの制御タスクの接続のプログラムコードは今回開発したJBGENで自動生成した．次に述べるHORBは異なる計算機間を結ぶI/Fプログラムコードの生成に用いられた．

2.2 分散オブジェクトの原理とHORBの特徴

従来，計算機同士をネットワークで結合し，物理的に離れた位置にあるプログラム間で通信を行うためには，通信手順(プロトコル)を設計し，その通信手順を実装するプログラムコードを記述していた．例えば，通信手順としては処理を依頼するコマンドとパラメータ，返事となるレスポンスとエラーコードなどが必要となる．複数の処理を依頼したい場合，コマンド体系は複雑になり，複数の処理の同時進行管理など，実装するプログラムコードも複雑になる．そのようなプログラムの設計は難しく，実運用が始まってから発見される潜在的な問題を抱えることがままあった．

オブジェクト指向プログラムでは現実世界の物事をオブジェクトというメタファーで抽象化し，オブジェクト間でメッセージを交換することで計算や処理を進める．図2-1に示すように，それらのオブジェクト間のメッセージのやりとりをネットワークを介して行えば，複雑になりがちな通信処理もオブジェクト指向の概念で統一して記述可能であるというのが分散オブジェクト(ORB)の基本的な考え方である．

そのような分散オブジェクト技術が実用化されたのは1990年代の中頃であり，OMG (Object Management Group)のCORBA(Common Object Request Broker Architecture)[9,16], Sun Micro SystemsのRMI (Remote Method Invocation)[17], MicrosoftのDCOM (Distributed Component Object Model)[18]，そして最近ではXMLを言語としてWeb Servicesを実現するW3CのSOAP(Simple Object Access Protocol)[19]がよく用いられている．これらは，企業の大規模な基幹業務用あるいは企業間商取引用というように，それぞれ想定する用途を有しており，適する利用分野や領域は大きく異なっている．

その中で，Javaを分散実行するHORB (Hirano's Object Request Broker)[1,20,21,22]は，初めて

異機種間で相互運用可能な分散オブジェクト処理系として 1995 年に発表され，これまで様々な研究開発プロジェクトや製品で利用されてきた．HORB はユーザが書いた Java プログラムを解析し，通信に必要なプログラムコードを自動的に生成する．通信手順の処理を明示的に記述する必要はなく，ユーザは生成されたプログラムコードを呼び出すだけで，ネットワークを介した通信処理を行うことができる．HORB は習得や利用が容易で，比較的小規模なハードウェアでも性能がよいといった特徴を有し，中・小規模のシステムに適している．

以下，HORB を用いたプログラムの実行モデルについて述べる．HORB の基本的な実行モデルは Proxy[23]である．Proxy は代理人という意味で，遠くの計算機上のオブジェクトの代理役を務める．プログラム開発時に，HORBC コンパイラを用いて，通信対象となるオブジェクト Or に対し，代理オブジェクトである Or_Proxy のプログラムコードを作成（自動生成）しておく．図 2-2 及び図 2-3 に示すように，実行時には通信を行いたいオブジェクト Os を含むプログラム Ps の内部に Or_Proxy のオブジェクトを生成する(図 2-3 の下の new の部分)．生成時には，HORB ランタイム・ライブラリとネットワークを介して，他方の計算機 B 上にオブジェクト Or が生成され，そのオブジェクトへの通信路が開設される．Or_Proxy は Or が有するメソッド（手続き，あるいは関数に相当）をすべて備えているが，メソッドの中身は通信処理であり，Or_Proxy のメソッドが呼ばれると通信が発生し，実際のオブジェクト Or の対応するメソッドが呼び出される．Or が返値を返すと，その値が Os に返信される(図 2-3 の下の greetings()を呼び出す部分)．このようにして，遠隔地にあるオブジェクトのメソッドをあたかも同じ計算機 A 内にあるオブジェクトのように呼び出すことが可能となる．

元々オブジェクト指向はシミュレーションのために発案された概念であり，本研究においても，人工衛星のセンサ，スラスタといった「物」をプログラム中でオブジェクトとして表現し，そのオブジェクト間で通信し合うという考え方によってシミュレータの設計を行っている．HORB ではプログラムにおいて通信処理のための記述量が最小限であるように考慮されている．例えばセンサであれば，センサのクラスを HORBC コンパイラにかけるだけでそのセンサを表す Proxy クラスが生成され，ネットワークを介した呼び出しが可能となる．また，Java インターフェース機能を用いてセンサの API(Application Program Interface)を定義しそれを HORBC コンパイラにかけることにより，同一プロセス内でのセンサの呼び出しとネットワークを介したセンサの呼び出しがほぼ同一の処理手順で記述可能となる．その際にネットワークを介さないプログラムと異なるのは，主にセンサのオブジェクトをどこに作るかを指定する箇所と，ネットワーク断に備えて例外事象を処理する箇所の二箇所となる．次章で示すように，衛星 OBS も含めて，多くのコンポーネントが HORB によりネットワーク上に分散可能となっている．

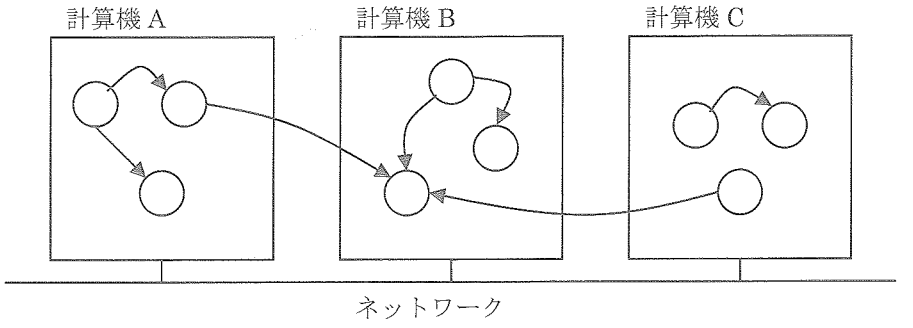


図 2-1 分散オブジェクトの概念図

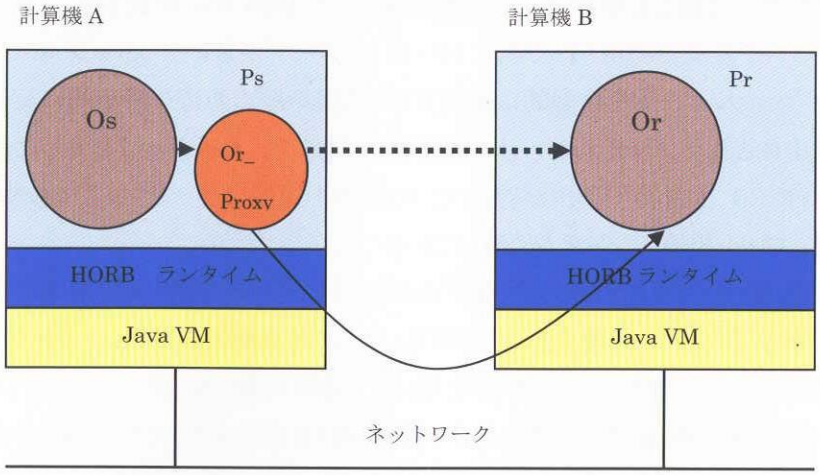


図 2-2 HORB によるプログラミングの概念図

```
Class Or {
    String greetings() {
        return "Hello!";
    }
}
```

```
Hello_Proxy remote = new Or_Proxy("horb://exp-obc.jaxa.jp/");
String s = remote.greetings();
```

図 2-3 呼ばれるオブジェクト(上)と呼ぶオブジェクトの一部(下)

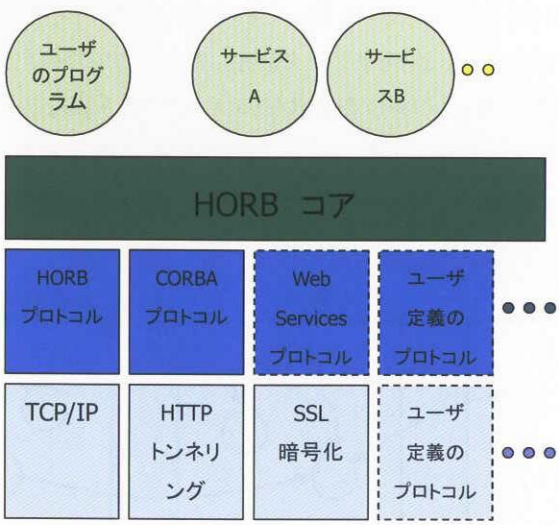


図 2-4 拡張可能なフレームワーク

このようなプログラム設計上の柔軟性に加えて，HORB は図 2-4 に示すように高い通信性能，非同期呼び出しなどのプログラミング能力を拡張する機能，CORBA IIOP との相互運用性，公開鍵認証によるセキュリティ機能などを備えている．また，利用者自身が通信プロトコルやサービスなどを動的に組み込むことができるフレームワークとしての性格を有することも大きな特徴である．従って将来，衛星専用バスやネットワークといった物理層上で業界標準を含む様々な通信プロトコルをに対応することも考えられる．注意すべき点としては，HORB は技術研究を目的として開発されているため，CORBA IIOP 実装が準拠する CORBA 仕様が少々古いなど，すべての機能が完全な状態にあるわけではなく，大規模な実用を行う際には幾分の整備を必要とする場合がある．

2.3 他ミドルウェア(CORBA, Web Services 等)との比較

多くの分散オブジェクト・ミドルウェアがある中で，CORBA, DCOM, SOAP(Web Services)はプログラミング言語に依存せず言語独立性を特徴とする．また，CORBA と SOAP はプラットホーム(OS や CPU)に依存しない．一方，DCOM は Windows のみで動作し，Java RMI と HORB は Java のみで動作する．Java 処理系は多くのプラットホームで利用可能であり，プラットホーム依存性は比較的低いと言える．ただし，HORB は Java 以外の言語もサポートしつつある．

一般に，プログラミング言語を特定するほど ORB はプログラミング言語の文法やセマンティクスと一体化され，プログラム内での利用が容易になる．また，速度やメモリ使用量といった性能も高い傾向がある．その点で HORB は最もプログラミング性がよく，性能がよい[22]．

逆に，言語独立性やプラットホーム独立性があるほどプログラミングは難しくなり，性能も犠牲となる．たとえば CORBA の場合は，プログラムとは別に独立のインターフェース定義言語 IDL(Interface Definition Language)で分散インターフェースを記述することにより，言語やプラットホームに対する独立性を得，不特定のベンダー間での相互運用性を実現している．その際は，IDL のセマンティクスとプログラミング言語のセマンティクスの違いへの配慮が必要となる．Web Services では多くの処理系で IDL に相当する WSDL(Web Services Description Language)を自動生成する機能を提供しており，プログラミングは容易になってきているが，性能は最も劣る．分散ミドルウェアを選択する場合，以上のトレードオフを勘案することになる．

本研究の場合，シミュレータの本体を Java で記述するため Java との親和性が必要であったこと，比較的高い性能が要求されること，搭載模擬計算機に組込む柔軟性が必要であったことから，HORB を選択した．HORB の CORBA IIOP プロトコルサポートやプロトコル・プラグイン機能により他の分散オブジェクト・プロトコルを使用する他機関の装置との相互運用性も視野に入れた．

図 2-5 に，2 台の計算機間におけるメソッド呼び出しの性能評価の結果を示す．横軸は一回のメソッド呼び出しに要した時間であり，グラフが短いほど高速であることを表す．縦軸は，種々の ORB とプログラミング言語の組み合わせである．上から 3 つは HORB を用いており，それぞれ，C#と Java 間，J#と Java 間，C#と J#間の通信を行っている(C#と J#については次節を参照)．4 つ目は代表的な CORBA 処理系である Java IDL を用いた Java と Java 間の通信，5 つ目は標準的な Windows.NET™ の Web Services を用いた C#と C#間の通信である．このグラフからわかるように，HORB は CORBA と比較して2倍程度高速であり，Web Services と比較して 10 倍程度高速である．

2.4 HORB のリアルタイム拡張と多言語対応

リアルタイム性が要求されるアプリケーションのために，HORB は下記のようなリアルタイム機能を有する．

- プライオリティシッピング

  クライアントからサーバオブジェクトのスレッド・プライオリティを動的に変更可能である．複数のクライアントのオブジェクトがそれぞれ異なるスレッドプライオリティで動作している場合，それらのクライアントが呼び出したサーバのオブジェクトも対応するプライオリティで動作しないとシステム全体としてのプライオリティに矛盾が生ずる．そのような事態が望ましくない場合，Or クラスを HORBC コンパイラで-rt オプション付きでコンパイルし，クライアントはサーバのオブジェクトのメソッドを呼び出す前に Or_Proxy._setPriority(int newPriority)を呼ぶことにより，指定のプライオリティを設定すればよい．

- スレッドプーリング

  サーバ上でサーバオブジェクトのスレッドを予めまとめて生成しておき，スレッド生成時間を削減する機能である．スレッド数の初期値と上限値を指定可能である．

- コネクションの再利用

  クライアント側で同一通信路上でのコネクション開設時間を削減するため，一度使用し終わったコネクションを削除せずに保存しておき再使用する機能である．

本研究では次章で述べるように，シミュレータと搭載模擬装置間の周期的な通信において，Java のガベージコレクションが発生し，周期に遅れが生ずる現象が観察された．この問題を解決するため，HORB 中のプロトコルドライバを改良し，バッファ等のメモリ領域を節約して通信遅延を抑制する対策を講じた．

一方，Java と並んで Windows.NET の C#, managed C++といった.NET 言語群が一般的に使用されるようになってきた．そこで，C#等，Java 以外のプログラミング言語で HORB を利用可能とするため，多プログラミング言語化の構想が進んでいる．HORB は大きく分けて Proxy 生成処理系(HORBC)とランタイム・ライブラリから構成されるが，Windows.NET 上では，J#言語という Java 言語と文法的に互換性のある言語が提供されており，J#へのランタイム・ライブラリの移植を行った．その結果，C#等の.NET 言語群と Java 間において基本的な HORB プロトコルによる通信が可能となった．当初は，搭載計算機で使用するために C++用の HORB 処理系の作成を検討したが，C++用の HORB の処理系はボランティアによってクライアント部分が移植され部分的には動作するが，今回の用途ではサーバ部分が必要であったために使用することができなかった．サーバ部分のランタイムは比較的複雑であり，これを作成することは課題となっている．今回は，搭載計算機に Java と μITRON のハイブリッド OS を用意し，Java 間の通信コードは HORB で，Java と C 言語で記述した μITRON タスク間の通信コードは新たに開発した Java-μITRON ブリッジ・ジェネレータ JBGEN によって生成した．

表 2-1 ミドルウェアの比較

	HORB	CORBA	HLA/RTI
実績	携帯電話 ロボット, …	広く企業で使われている	軍事, 航空宇宙
搭載機器との通信	○	○	×
分散化の容易さ	◎	○	△
通信速度	◎	○	△
サイズ	小	中	大
言語依存性	△(Java ただし IIOP 対応)	◎	△(Java/C++)
OS 依存性	◎(Java)	○	△
制約		IDL の知識が必要	HLA の知識が必要

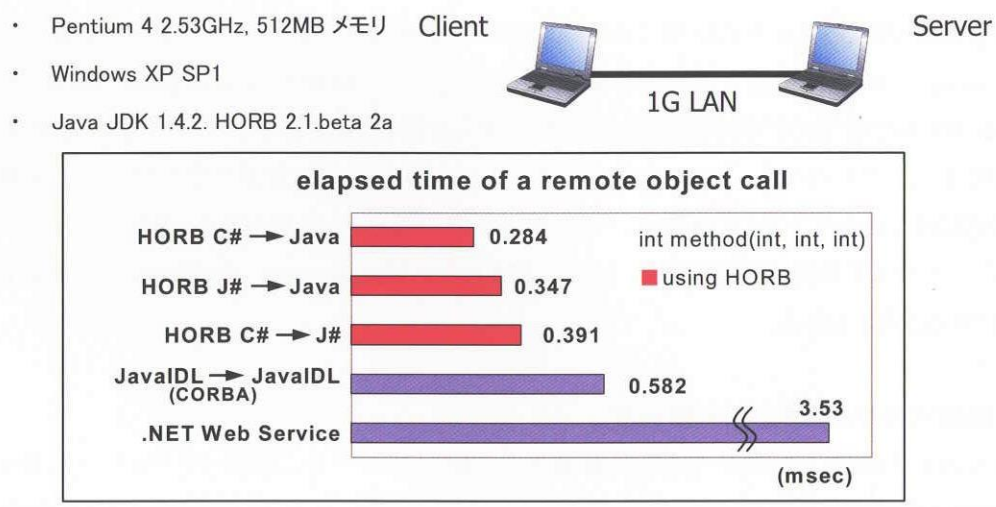


図 2-5 ORB 間の性能評価結果

3. 搭載ソフトウェアと宇宙機シミュレーション環境間の通信方式の検討

従来，宇宙機開発において OBS の開発・検証環境は概して図 3-1に示す仕組みで行われてきた．OBS はハードウェアや通信方式に依存した形で開発されることが多く，移植性や再利用性に乏しいため，衛星ごと用途ごとに形態が変化するシミュレータを容易に整備することが難しかった．OBS やシミュレーション・モデルは今後複雑化すると予想されるのに対し，信頼性を落とすことなく短期開発が求められる状況を踏まえ，本研究では以下を考慮して OBS 開発・検証環境を構築した．

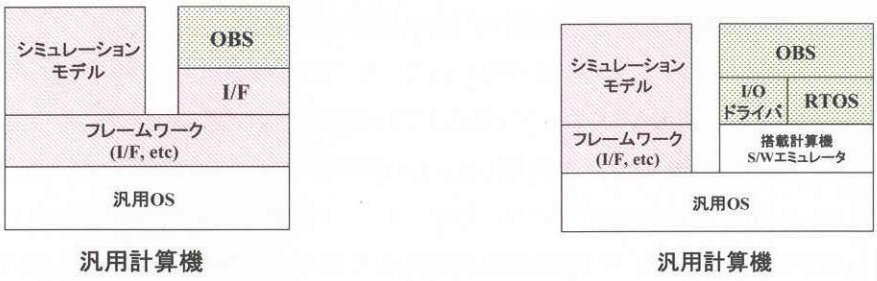
(1) ハードウェアの違いを吸収する OS や VM(Virtual Machine)の利用
(2) 通信方式の違い(TCP/IP, MIL-1553B, 等)を吸収するミドルウェアの利用
(3) 可能な限りプログラムコードを自動生成

具体的には，(1)には国産 RTOS である μ ITRON と JavaVM，(2)には HORB，(3)には MATLAB/Simulink™と本研究で開発したツール JBGEN(3.4節参照)を用いた．このようなアプローチは，OS, VM, ミドルウェア等が信頼性，拡張性ともに高いことが前提となるが，OS に関しては他分野で実績のあるリアルタイム OS である μ ITRON を信頼性を検証した上で JAXA 開発 MPU(Micro Processing Unit)上に搭載する計画が進められている．また HORB は Java 言語で開発される組み込みシステムではすでに広く利用されている．これらを用いることでシミュレーション・モデルや OBS の開発はそれぞれのロジックにのみ集中でき，モデルの再利用が容易になることから，開発期間の短縮が期待できるだけでなく無駄なバグの混入が防げるという利点がある．また，シミュレータを利用したソフトウェア検証が容易になり，JAXA では IV&V (Independent Verification & Validation)に応用できると考えられる．

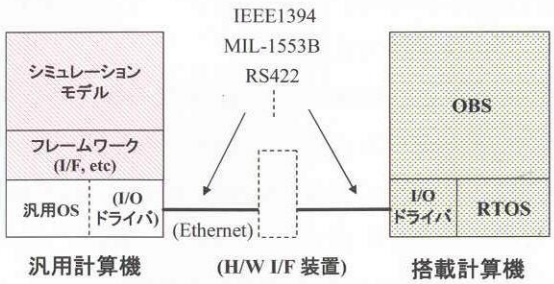
3.1 搭載ソフトウェア検証用実験装置

図 3-2は，HORB を利用して搭載計算機を宇宙機シミュレータに接続する PILS の実現性を確認するために整備された実験装置の構成である．ここでは OBS として C 言語で記述され搭載計算機上の μ ITRON 仕様 RTOS 上で動作する制御タスク等を想定した．搭載模擬装置には，μ ITRON3.0 仕様[10]の RTOS に Java 実行環境を統合した JTRON1.0 仕様[11]の J-right/V™ [12]を使用した．搭載模擬装置側に置かれた Java I/F 等は，シミュレータ側から μ ITRON タスクが通常の Java オブジェクトと同等にみなすことができ，また HORB を用いて相互通信を容易に可能にするためのものである．これは，Java 言語で短い I/F を記述すれば，HORB および JBGEN により仕様から自動生成される．

μ ITRON-Java 間の通信に μ ITRON の機能であるメイルボックス(mailbox)を使用することで，OBS は通信デバイスに依存せずに記述できる．また，搭載模擬装置とシミュレータ間の通信に HORB のフレームワークを利用することで，通信方式の違いが吸収可能である．従って，開発者はシミュレーション・モデルと OBS それぞれのロジックに集中して開発を行うことが可能となり，移植性や再利用性が高まると考えられる．



(a) 汎用計算機上での OBS 開発・検証　　(b) S/W エミュレータを用いた OBS 検証



(c) シミュレータと連係した搭載計算機検証

図 3-1　従来の搭載ソフトウェア開発・検証環境

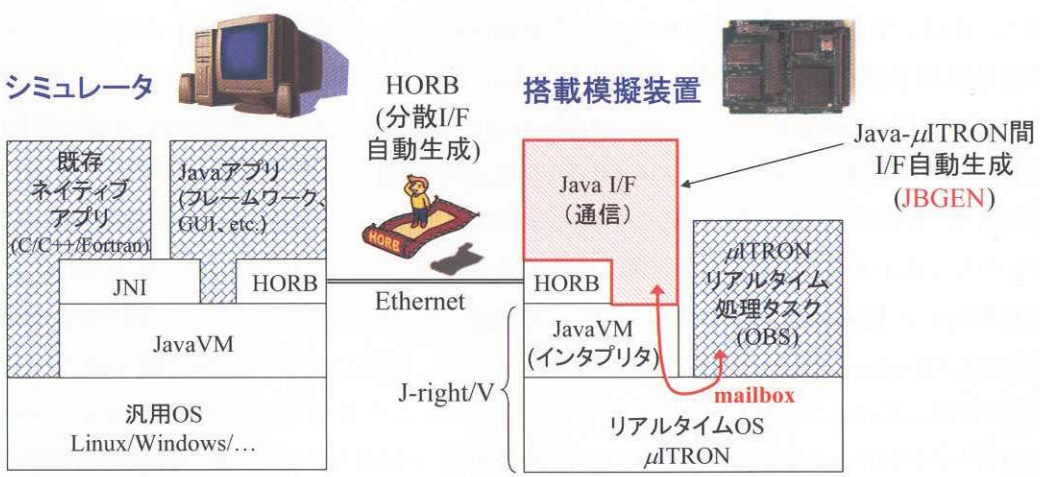


図 3-2　HORB を用いた搭載ソフトウェア検証実験装置

3.2 通信実験

3.2.1 HTV の簡易 AOCS タスクを用いた開ループ試験

本実験は，JBGEN 開発前に行われた基礎実験である．その目的は，以下の 3 つにまとめられる．

(1) MATLAB/Simulink で開発されたモデルの μ ITRON への移植性確認．

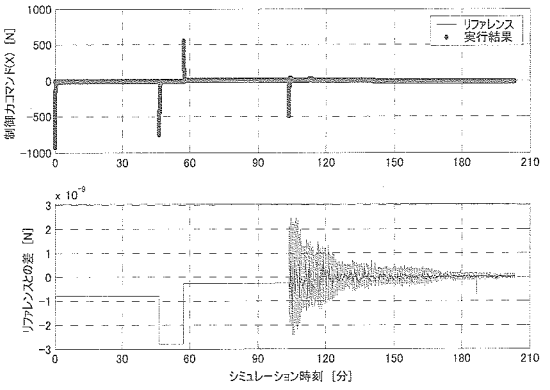
(2) 搭載模擬装置上の μ ITRON と Java 間の通信 I/F の設計と動作確認．

(3) HORBを用いた搭載模擬装置と汎用PC間の通信性能確認.

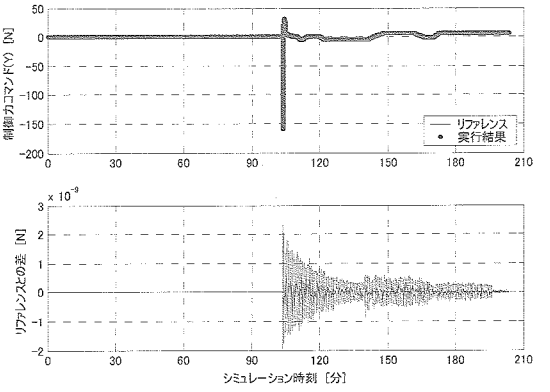
これらに対して,実験から次のような結果が得られた.まず(1)を確認するため,搭載模擬装置上のリアルタイム・タスクとして,通信ミドルウェアHLA/RTI評価のために作成した簡易HTV/ISSシミュレータ[2]のHTV制御系簡易モデルを利用した.この制御系モデルはMATLAB/Simulink上で作成されており,Real-Time Workshop™(RTW)のオートコード機能を用いて生成されたCソースコードに対して,μITRON上で8Hzの周期起動タスクとして動作させるための改修および機能追加を行った.この改修,機能追加は非常に少ないため,RTWにより生成されたCソースコードはほぼそのまま流用することが可能であることが確認できた.したがって,μTRON仕様で書かれた実際のOBSも本構成に適合させることが可能と考えられる.

次に(2)について,搭載模擬装置上の制御タスクとJava間の通信I/Fは,JBGEN開発以前にハンド・コーディングで作成した.その通信方式は,μTRONのメイルボックスを用いたデータ送受信(バイナリ形式)により行い,JBGEN開発の基礎となった.まず,搭載模擬装置側の通信I/Fは,シミュレータ側から渡された入力データを送信メイルボックスに格納し,受信メイルボックスに値が格納されるまで待機する.制御タスクはその起動周期ごとに送信メイルボックスの中身を確認し,新規のデータがある場合はそれを入力として計算して結果を受信メイルボックスに格納する.新規データが無い場合は,何も行わないものとした.最後に搭載模擬装置側の通信I/Fは,受信メイルボックスの内容をシミュレータ側に返す.この一連のプロセスがシミュレーションの間繰り返される仕様とした.

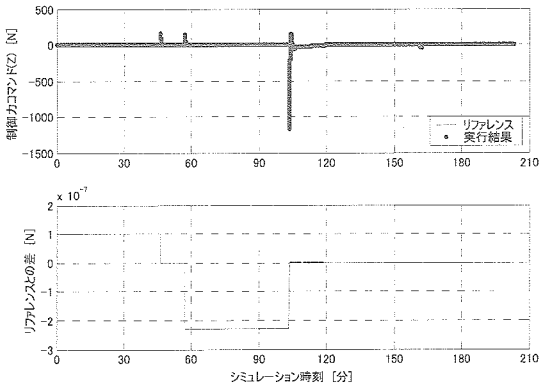
最後に,作成したHTV制御系簡易モデルの制御タスクとJava間の通信I/Fを用いて,シミュレータと搭載模擬装置間の通信性能を確認するためにAOCSの開ループ試験を行った.本試験は,予めファイルに書き込まれたセンサ・データをシミュレータが読み込んで搭載模擬装置に送信し,搭載模擬装置で制御力トルク・コマンドを計算しシミュレータ側に返すシーケンスを繰り返すことにより行われた.本試験において,AOCSタスクで計算されシミュレータ側に返された制御力トルク・コマンドと予めHLA/RTIの評価時に計算しておいたリファレンス・データとの差を図3-3に示す.また,通信呼び出し間隔をシミュレータ側(Linux)で計測した結果を図3-4に示す.呼び出し間隔は理想的には125msec(8Hz)で一定でなければならないが,実際には数分間隔で数100ミリ秒の通信遅延が発生していることが分かる.3.3節で述べるように,この通信遅延の原因はJavaのメモリ管理機能(ガーベージ・コレクション: GC)によるものであることが明らかになった.搭載模擬装置側のJava通信I/Fやシミュレータ側のプログラムにおいて,ローカルにオブジェクトを生成しないように改修してもGCによる通信遅延が消えなかったことから,HORBの改修を行うことで対処した.ただし,搭載模擬装置側では第一世代のJavaVM(Sun JDK1.1)を使用しているため,GCのアルゴリズムも旧式であり,プログラムがポーズ状態に陥りやすいことは既に明らかで第二世代のJavaVM(Sun J2SDK 1.2以降)であれば,GCによるプログラムの一時停止が起こりにくいインクリメンタルGC等を利用することができる.またリアルタイム仕様のJava(RTSJ)を利用すれば,リアルタイム性の保障も可能である.したがって,今回のHORBの改修は応急措置であり,最新のJavaVMの利用により問題は解消できると考えられる.



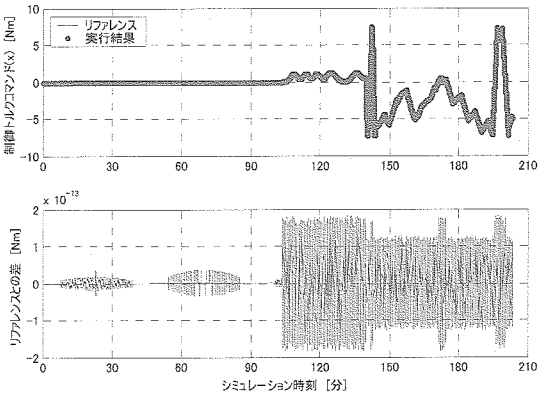
(a) 制御力コマンド(x 方向)



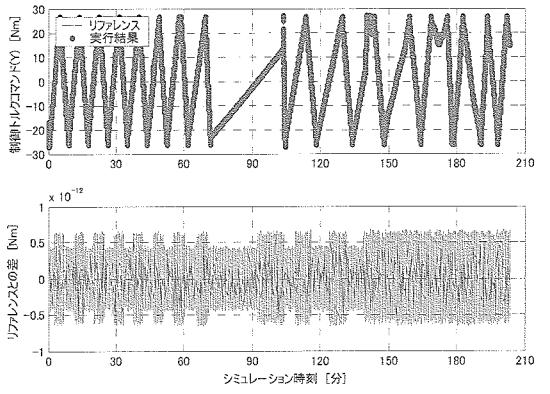
(b) 制御力コマンド(y 方向)



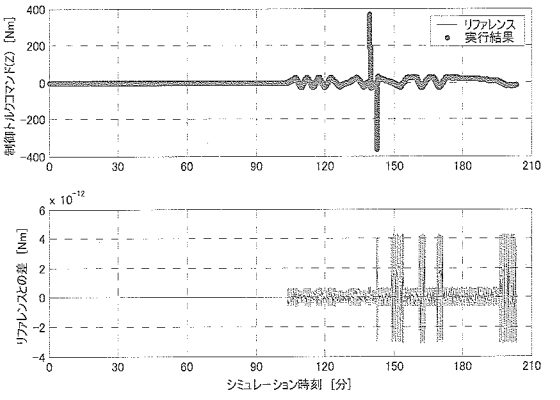
(c) 制御力コマンド(z 方向)



(d) 制御トルク・コマンド(x 方向)



(e) 制御トルク・コマンド(y 方向)



(f) 制御トルク・コマンド(z 方向)

図 3-3　制御力／トルク・コマンドと誤差

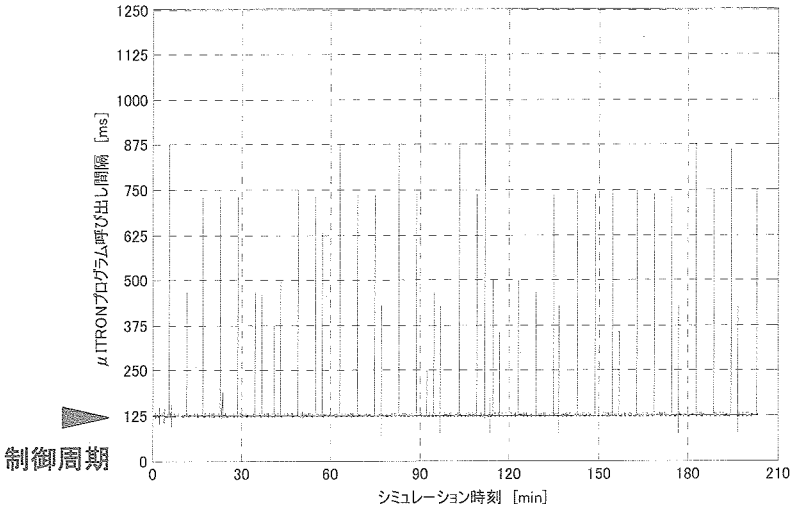


図 3-4 通信遅延の発生

3.2.2 HTV/ISS の簡易シミュレータを用いたマルチタスク閉ループ試験

本実験では図 3-5に示すように，HORB により容易に分散化可能なように一部改修した HTV/ISS の簡易シミュレータを用いて閉ループ・シミュレーションを行い，FSS と PILS で共通な I/F の実現性，および分散シミュレーションの実行性能を評価した．

シミュレーションのシナリオは，HTV の R-bar 接近フェーズであり，ISS の 100m 手前で 180 度ヨー・マヌーバし待機状態に入る．コマンドにより接近を再開して再度 30m 手前で待機状態となり，さらにコマンドにより 10m 手前まで接近を進めることができる．100m 手前より接近している状態では，アボート・コマンドによりいつでも ISS から退避できる．図 3-6に，シミュレーションにおける HTV と ISS の様子を 3D 表示するモニタ(Java 3D)と，簡易的に地上局を模擬したテレメトリ表示・コマンド送信用のパネルを示す．

OBS は，3.2.1 項で用いた AOCS タスク(8Hz)に TTC (Telemetry, Tracking & Command Subsystem)タスク(1Hz, 0.1Hz, 即時応答)を追加することによりマルチタスク化した．表 3-1に本実験で扱ったテレメトリおよびコマンドを示す．TTC タスクの機能は以下の通りである．

(1) HTV/ISS 簡易シミュレータのデータを収集する．
(2) テレメトリおよびコマンドを CCSDS (Consultative Committee for Space Data System)パケットのフォーマットに従って生成する．また，受信した CCSDS パケットからデータの復元を行う．
(3) CCSDS パケットの宛先に応じてパケットの転送を行う経路制御．

AOCS, TTC の両タスクは，FSS 用の Java プログラムと PILS 用にμITRON 上で動作する C プログラムの両方を整備した．また FSS と PILS に共通に用いられるシミュレータからタスクを呼び出す I/F の仕様を検討した．用いた共通 I/F を図 3-7に示す(詳細は 3.4 節参照)．PILS の場合はこれを HORB および JBGEN の入力としてシミュレータと搭載模擬装置間，およびμITRON-Java 間の通

信 I/F を自動生成した．本実験で自動生成したコードは，図 3-8に示すように MATLAB/Simulink モデルから生成した AOCS タスク，センサ／アクチュエータ・モデル，HORB により自動生成されたシミュレータと搭載模擬装置間の分散 I/F と JBGEN により生成したμITRON-Java 間通信 I/F である．これらを自動生成することにより，作業が極めて効率化できることが確かめられた．

さらに自動生成された I/F 群と HORB が介在する PILS においても，FSS と同様のシミュレーションが容易に実行でき，またμITRON 上の周期の異なる AOCS および TTC タスクはそれぞれ正常にシミュレータと通信することを確認した．

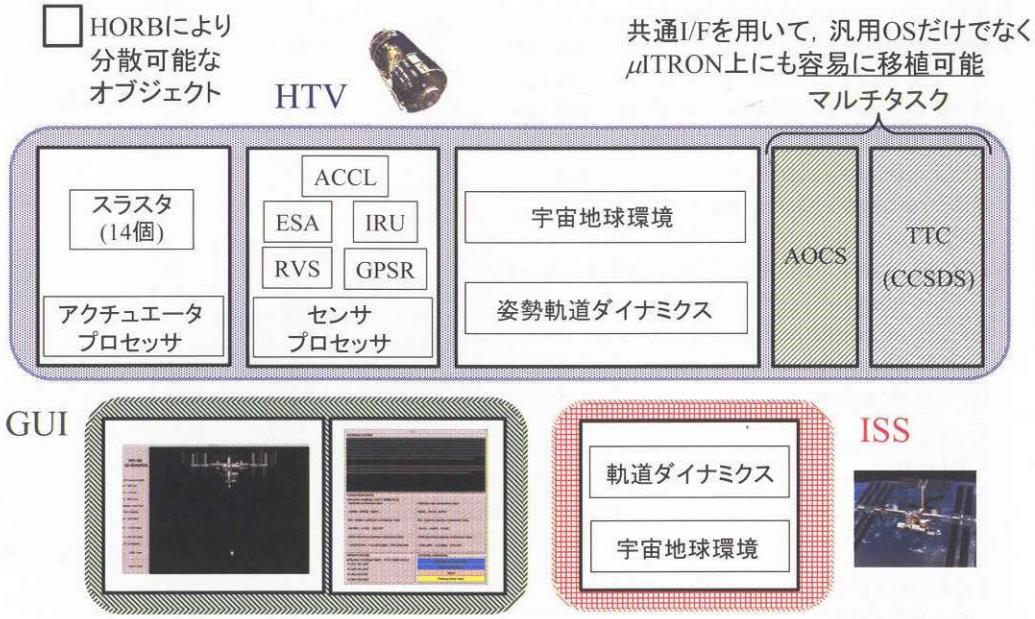
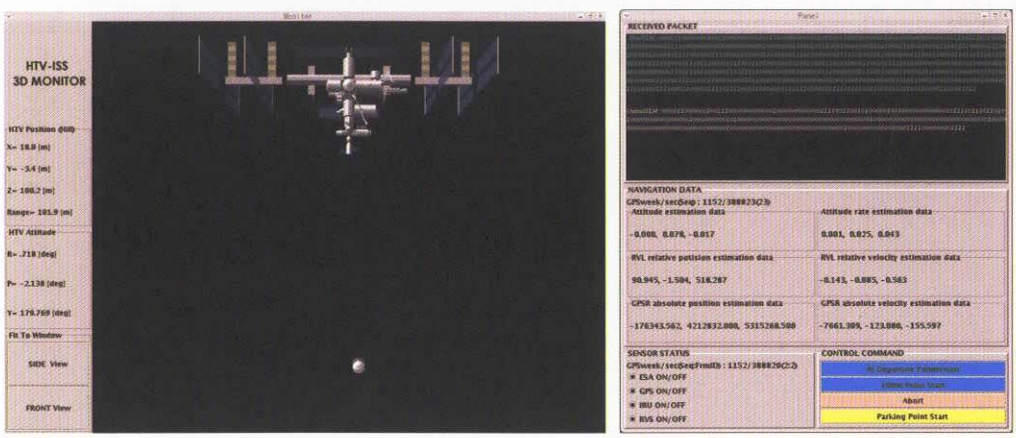


図 3-5　HTV/ISS 簡易シミュレータの構成



(a)3D 表示用モニタ　　(b)テレメトリ表示・コマンド送信用パネル

図 3-6　HTV/ISS 簡易シミュレータの GUI

表 3-1 テレメトリおよびコマンド

Item	Name	Data Length [bit]	Rate [Hz]	From	To
Navigation data	Counter	32	1.0	AOCS	Panel
	Attitude estimation data (roll)	32	1.0		
	Attitude estimation data (pitch)	32	1.0		
	Attitude estimation data (yaw)	32	1.0		
	Attitude rate estimation data (roll)	32	1.0		
	Attitude rate estimation data (pitch)	32	1.0		
	Attitude rate estimation data (yaw)	32	1.0		
	RVS relative position estimation data (x)	32	1.0		
	RVS relative position estimation data (y)	32	1.0		
	RVS relative position estimation data (z)	32	1.0		
	RVS relative velocity estimation data (v_x)	32	1.0		
	RVS relative velocity estimation data (v_y)	32	1.0		
	RVS relative velocity estimation data (v_z)	32	1.0		
	GPSR absolute position estimation data (x)	32	1.0		
	GPSR absolute position estimation data (y)	32	1.0		
	GPSR absolute position estimation data (z)	32	1.0		
	GPSR absolute velocity estimation data (v_x)	32	1.0		
	GPSR absolute velocity estimation data (v_y)	32	1.0		
	GPSR absolute velocity estimation data (v_z)	32	1.0		
Status	Counter	32	0.1	Sensor	Panel
	IMU data	32	0.1		
	ESA status	32	0.1		
	GPSR	32	0.1		
	RVS	32	0.1		
	ESA ON/OFF	1	0.1		
	GPS ON/OFF	1	0.1		
	IRU ON/OFF	1	0.1		
	RVS ON/OFF	1	0.1		
Command	AI departure permission	1	~	Panel	AOCS
	100m point start	1	~		
	30m parking point start	1	~		
	Abort	1	~		

※ Panel:テレメトリ表示・コマンド送信用パネル(GUI)を示す

```
public interface AOCS extends JBinterface {
    /**
     * @arraysize   input   31
     * @arraysize   return   33
     */
    public double[] advanceAOCS(double[] input) throws JbException;
}
```

(a) AOCS 用インターフェイス

```
public interface DataHandler extends JBinterface {
    /**
     * パケットの送出を行う
     * @arraysize return   128
     * @return   送信 CCSDS パケット
     */
    public short[] getPacket() throws JbException;

    /**
     * パケットの受信を行う
     * @arraysize packet   44
     * @param packet      受信 CCSDS パケット
     */
    public void setPacket(short[] packet) throws JbException;
}
```

(b) TTC 用インターフェイス

図 3-7　FSS/PILS 共通インターフェイス

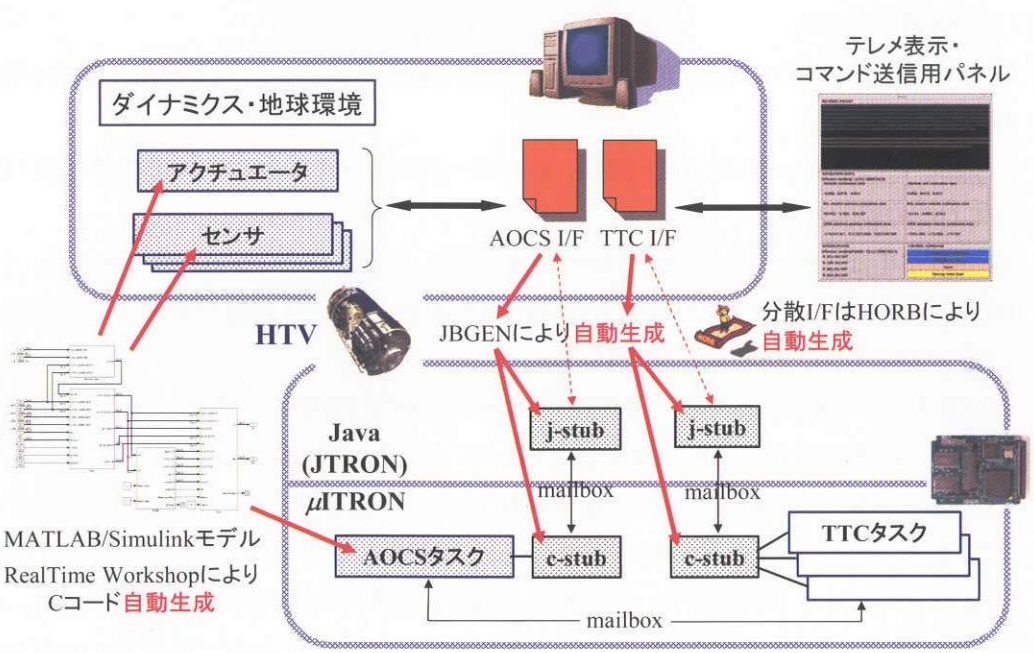


図 3-8　共通インターフェイスと自動生成コード

3.3 通信遅延の解消

シミュレータは搭載模擬装置の AOCS タスクを毎秒 8 回(125msec 間隔)呼び出す. 図 3-4で示したように, 約 5 分毎に 125msec を大きく越える通信遅延が発生し, リアルタイム性が損なわれていた. 本節では実験的に行った HORB を改修して遅延を解消する試みについて述べる.

3.3.1 通信遅延の原因の特定

プログラマが自らオブジェクトの生成(new)と消去(delete)を行う C++と異なり, Java では使わなくなったオブジェクトを明示的に消去する必要はなく, メモリ管理は JavaVM によって自動的に行われる. JavaVM はオブジェクトを生成する際に, オブジェクト用のメモリ領域であるヒープメモリからメモリを割り付ける. その際にヒープメモリが足りなくなると, 使われなくなったオブジェクトを探索・破棄してメモリ領域を回収する. この処理はゴミ集め(ガベージコレクション, GC)と呼ばれる. C++や C ではメモリ解放(delete あるいは free 操作)を忘れるとメモリリークが発生し, 長期運用中にメモリを使い尽してシステムが予測不能な状態に陥る危険があるが, Java ではその危険性は格段に減っている. 大規模な基幹システムで Java が使用されるのは, その信頼性に依るところが大きい.

しかし, GC はプログラマが予期しないタイミングで発生するために, 組込みシステムにおいてリアルタイム性が必要な場合は何らかの対策が必要である. Java のリアルタイム拡張仕様[24]ではメモリ領域をオブジェクトの寿命によって分類して使い分けたり, プログラムから GC の実装に GC のタイミングに関する指示を出すための API を定義している. このリアルタイム拡張仕様はまだ一般的に利用可能ではないため, 本研究では他の手段によってリアルタイム性の確保を試みた.

前節での解析の結果, 搭載模擬装置の Java 処理系で GC が発生すると, HORB を用いた通信が GC の間待たされ, 800msec 程度の大きな遅延が発生することが明らかになった. そこで考察の結果, 以下の仮説を立てた.

(1) GC は, AOCS タスクの周期的な呼び出しの度に行われるオブジェクトの頻繁な生成と消滅に起因する.

(2) J-Right/V™ の JavaVM は, GUI 画面, ネットワーク通信の度に常にメモリを消費し続ける. 従って GC の原因となる.

(3) JBGEN が生成したコードは AOCS タスクの呼び出しの度にはオブジェクトを生成しない. また, HORB が生成した AOCS タスク呼び出しのコード中でも呼び出しの度にはオブジェクトを生成しない. 従って GC の原因とならない.

(4) AOCS タスクの呼び出しの度に, HORB ランタイムライブラリ中でオブジェクトの生成を行っている. 生成されたオブジェクトは使い捨てられているため, GC の原因となる.

まず, (2)のシミュレータを動作させない定常状態でのメモリ消費量を調べた結果, J-Right/V の JavaOS は 4 分ごとに 96 バイトを消費していた. そのペースが続くとすると, 20MB のヒープメモリの場合, 580 日以内に GC が発生する. 長期運用の場合, 定期的に意図的な GC を起こして, メモリをクリアする必要がある (System.gc()を使用). (3)の JBGEN が生成したコード(AOCS_Impl.java)及びランタイムライブラリ(Jbgen.java), そして HORB が生成したコード(AOCS_Skeleton.java)は, 接続時にはメモリを使用するが, メソッド呼び出しの度に新しくオブジェクトを生成することはない. 従っ

て，125msec 毎に呼び出される advanceAOCS()の度にメモリを消費しているのは(4)の HORB のランタイムライブラリと考えられる．そこで HORB に省メモリ化の改良を施し、通信遅延の解消を試みた。

3.3.2 省メモリ化の効果の測定結果

省メモリ化した HORB を用いて，ヒープメモリ量を 20MB とした J-Right/V の advanceIOCI()を呼び出す実験を行った．実験の条件は以下の通りである．

- 都合により図 3-4の計測をした機種とは違う計算機上で J-Right/V を動作させた．クロック速度はどちらも 300MHz 程度，ヒープメモリ量はどちらも 20MB 程度．
- 省メモリ化の効果が明確になるように，JBGEN と ITRON タスクは使用せず，advanceAOCS()のメソッドの中身は 125msec スリープ後に double[]の配列を返すだけとする．

遅延時間の推移を図 3-9に示す．図 3-1と比較して，遅延の間隔は 5 分おきから 10 分おきへ，一回の遅延の大きさは 750msec 程度から 300msec 程度に改善された．しかし，依然として遅延は発生している．ヒープメモリの空き状態を観察したグラフが図 3-10である．ヒープメモリが減ってゆき，ゼロに近くなると GC が発生してヒープメモリの空きが増え，同時に通信遅延が発生していることがわかる．原因は J-Right/V の Java クラスライブラリのバッファ書き出しメソッド flush()が呼び出しの度にメモリを消費していることである．これは，次に示すように新しいバージョンの J-Right/V(JBlend™)を導入することで解決すると考えられる．

図 3-11に，同様の実験を新しい JavaVM である JDK1.4 上で行った結果を示す．J-Right/V ではメモリを消費した flush()が JDK1.4 では消費しないよう改良されている．定期的な通信遅延は起きておらず，JDK1.4 では省メモリ化の効果があることがわかる．125msec よりも下に振れている線があるが，これは Windows のタイマーの精度が 10msec 程度と荒く，量子化誤差が出ているためである．125msec より上に振れている線は，「ウィルスバスター　パターンファイルのアップデート」といった他のプロセスの影響と考えられる．

そこで本節では以下の通り結論する．

- HORB の省メモリ化により，最近の JavaVM を使用すれば，周期的なリモートメソッドの呼び出しは通信遅延を起こさないことが実証された．
- 周期的なメソッド呼び出しではメモリは節約されるが，通信路の開設時やエラー発生時にはメモリを消費する．そのため，定期的に意図的な GC を行い周期的なメソッド呼び出し時に GC が発生しないようにする必要がある．

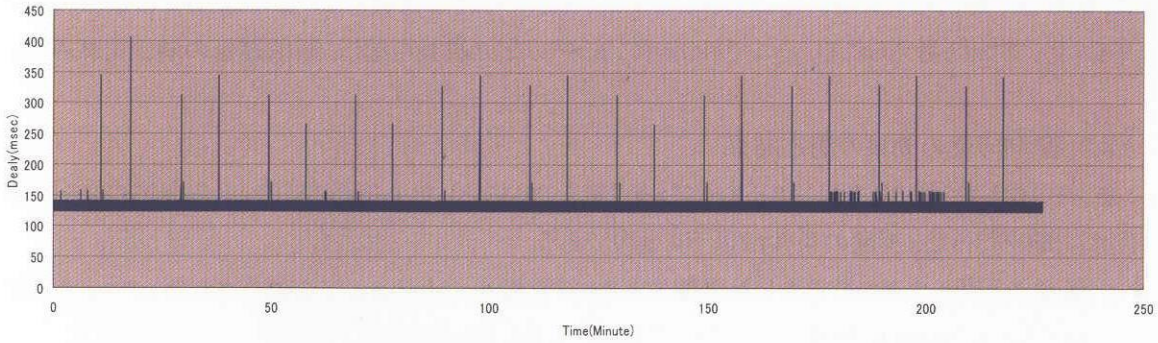


図 3-9 改善された J-Right/V での呼び出し時間

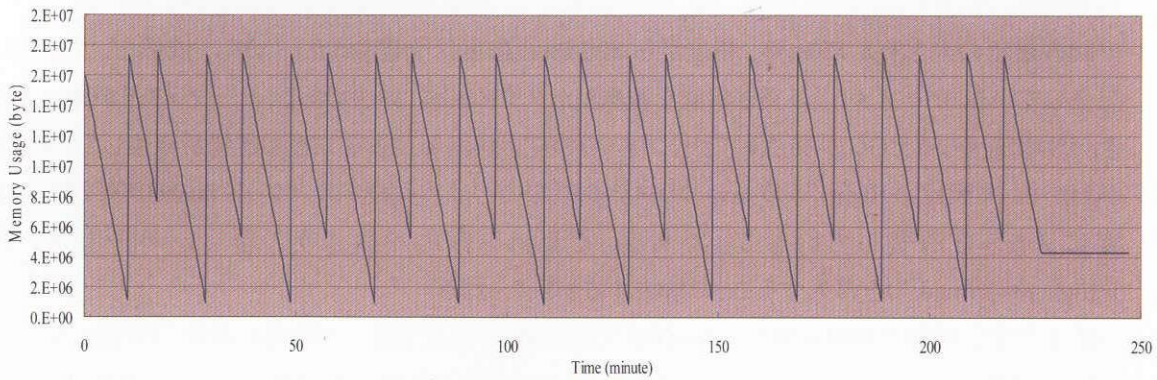


図 3-10 J-Right/V でのヒープメモリの空き状態の遷移

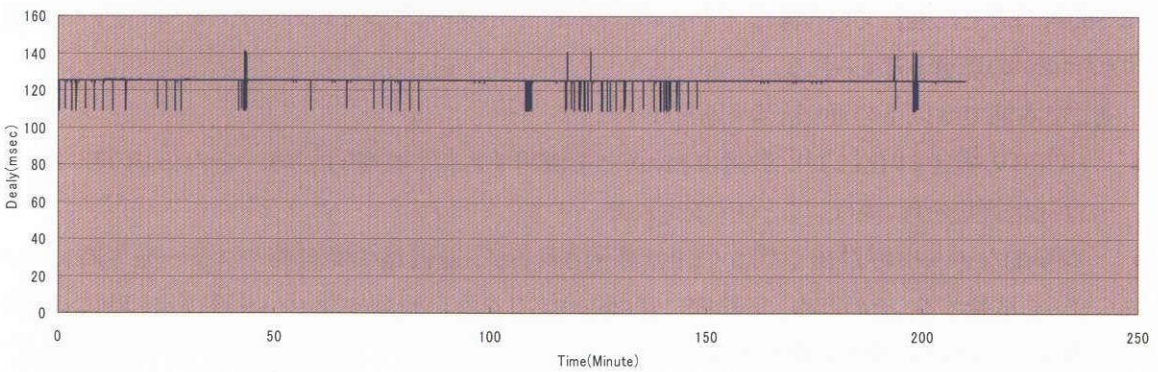


図 3-11 遅延が解消されている JDK1.4 での呼び出し時間

3.4 Java-μITRON ブリッジ・ジェネレータ JBGEN

ここでは，本研究で開発した搭載模擬装置上の Java プログラムと制御タスク(ITRON タスク)間の通信のプログラムコードを通信の仕様から生成するジェネレータについて述べる[25]．プログラミング作業時間は事実上ゼロになり，開発期間が短縮されるとともにプログラムミスが混入する機会が大幅に減少する効果を得た．

3.4.1 目的

本研究の搭載模擬装置で使用したオペレーティング・システムである J-Right/V は，μITRON 上にインタプリタ式の小型 JavaVM を搭載するアプリックス社の JBlend を，パーソナルメディア社が x86 プロセッサの PC に移植したものである．元になる JBlend は Java 搭載(i アプリ等)携帯電話や情報機器などで大量に使用されており，市場で大きなシェアを有している．J-Right/V と JBlend では，ITRON 上のタスクはもとより，Java のスレッドのひとつずつが ITRON のタスクとなっている．ITRON 上の Java が ITRON のシステムコールを呼び出すための標準規格が，TRON 協会により JTRON 規格として定められている．J-Right/V は JTRON 1.0 API を提供しており，Java スレッドがタスクの生成やメモリの割り当てを行ったり，Java スレッドと ITRON タスクとの間でタスク間通信を行うことができる．Java のガベージコレクション・スレッドは最も優先度が低い ITRON タスクとして動作する．リアルタイム性が高い処理は ITRON で，通信やユーザインターフェースなどリアルタイム性をあまり必要としない処理は Java で行うといった使い分けが可能である．

さて，図 3-2に示す実験装置上において，前節で述べた試験のシミュレーション・ループは以下の手順で構成されている．シミュレーション・ループは，AOCS と TTC の 2 系統あり，それぞれが独立して動作する．

(1) Linux マシン上のシミュレータ本体の Java プログラムが HORB を用いて搭載模擬装置の Java プログラムのメソッドを呼び出す．シミュレーションのループは 2 系統あり，それぞれメソッドの呼び出しインターフェースは図 3-1にある通りである．
(2) 搭載模擬装置の Java プログラムは JavaVM が備える JTRON API を介して，μITRON のメールボックスにコマンドとデータを送る．
(3) C で記述された ITRON タスクはメールボックスからコマンドとデータを受け取り，AOCS タスクあるいは TTC タスクの処理を行う．
(4) ITRON タスクは返値を返信するために，メールボックスにステータスと返値を格納する．
(5) 搭載模擬装置の Java プログラムはメールボックスからステータスと返値を受け取り，HORB のリモートメソッド呼び出しの返値としてシミュレータ本体の Java プログラムに値を返す．

このうち，(1)と(5)は HORB によってプログラムコードが生成される．(2)から(4)までは，これまで手作業でコーディングを行っていた．この通信処理は定型的であるが複雑であるため，潜在的なプログラムミスの原因となる．そのプログラミングを自動化するため，Java プログラムと ITRON タスク間でメールボックスを用いて通信するコードを生成するブリッジ・ジェネレータ JBGEN を開発した．

JBGEN によって，以下の効果が期待される．

(1) 複雑で定型的な処理が自動化されるため，大幅なコスト削減と信頼性の向上が期待できる．
(2) Java と C の間の役割分担の設計と変更が容易になる．
(3) HORB を同時に用いることにより，ネットワーク上の他の計算機から ITRON タスクをあたかも同じプログラム中にあるかのごとく呼び出すことができる．

3.4.2 JBGENの構成，通信仕様の与え方と出力ファイル

JBGEN は図 3-12，図 3-13のように，行いたい通信の仕様を入力すると，その仕様に従った通信処理を行うJava側のプログラムコード(スタブ)とC側のプログラムコード(スタブ)を出力するプログラム生成系である．JBGEN はジェネレータ，Java 側ライブラリ，C 側ライブラリから構成される．通信仕様の入力の文法とセマンティクスは Java インターフェースとして与える．インターフェースは複数のメソッドを含んでおり，Java 側は ITRON タスクをそのメソッドを実装するオブジェクトとみなして呼び出す．従って，生成される Java 側スタブはそのインターフェースを実装するクラスとなる．ITRON タスクでは，メソッド名に対応する関数を用意しておき，JBGEN のランタイム・ライブラリによってJavaからCに型変換されたパラメータを受け取って処理を行う．

ジェネレータの入力

- Java インターフェイス定義
  Java プログラムと ITRON タスクの通信内容をメソッドとして定義するインターフェイスである．図 3-12では Sample.java となる．メソッドのパラメータが配列を含む場合，C 側スタブでの配列サイズの最大値を与える必要があるため，配列の最大サイズを javadoc コメントとして指定する．図 3-7の@arraysize が配列サイズの指定である．これにより，実行時の配列サイズエラーの検出が可能となり，信頼性が向上している．
- Java-C 間のデータ型変換テーブル
  Java では基本データ型のビット長，符号の有無，精度は厳格に定義されており，機種によらない．C ではそれらは機種依存であり，コンパイラによっても異なっている．様々な機種に対応可能とするため，JavaとCのデータ型のマッピングは開発時に変換表として与えることができる．

ジェネレータの出力

- Java 側スタブ
  入力インターフェイス定義の実装クラスである．Java で記述され，Java インターフェイスファイル 1 つに対して 1 つが対応する．クラス名はデフォルトとして入力インターフェイス定義のインターフェイス名に「_Impl」を付加したものとする．図 3-12では Sample_Impl.java となる．HORB を用いてリモートから利用する場合，この Java 側スタブは，リモートの Java プログラムによって Sample_Proxy を介して使用される．JTRON では，Java のスレッドはそれぞれ ITRON のスレッドになっている．Java 側スタブは，それぞれスレッドとして動作する．従って，Java 側スタブと下記 C 側スタブとは JTRON が提供するメールボックスによって，ITRON スレッド同士の同期を行いながらデータの受け渡しを行う．
- C 側スタブ
  Java 側スタブとの間で ITRON を通じてデータの受け渡しを行うモジュールである．C で記述され，Java 側スタブファイル 1 つに対して 1 つが対応する．ファイル名は入力インターフェイス名に「_CStub」を足したものとする．図 3-12では Sample_CStub.c がこれにあたる．Java 側スタブとメールボックスによってデータの受け渡しを行うための関数を ITRON タスクに提供する．また，関数や定数を宣言するヘッダファイル Sample_CStab.h を生成する．

入力インターフェース定義中のひとつのメソッド名を getSensor()とすると，getSensor()用の C 関数は以下が定義される．

- ✓ getSensor_waitConnectItronTask() Java 側からの接続を待つ
- ✓ getSensor_input() メソッド呼び出しを待ち，パラメータを受けとる
- ✓ getSensor_output() メソッド呼び出しの返り値を返す

Linux 計算機上のシミュレータ本体が AOCS_Proxy オブジェクトを生成すると HORB と JBGEN を介して搭載模擬装置の ITRON タスクである AOCS タスクと接続される．シミュレータ本体が AOCS_Proxy.advanceAOCS()を呼び出すと，AOCS タスクは AOCS_CStab.c が提供する advanceAOCS_input()等の関数を使用して Java 側からのコマンドとパラメータを受け取り，advanceAOCS_output()を使用して結果を返す．その結果は返値としてシミュレータ本体に返信される．

3.4.3 実装と試験

実装においては，Java のシグニチャ(メソッドの型情報)の取得に javadoc API を用いた．Java が備える javadoc API は，プログラムのコメントに仕様情報を記述しておき，API 仕様書を生成する機能である．メソッドの型情報を得る際に通常用いられるリフレクションではなく，javadoc API を用いたのは，シグニチャの取得と同時に配列の最大サイズを指定するために今回追加した javadoc コメント である@arraysize の処理も行うためである．JBGEN が生成するコードの信頼性の確保のため，基本型の転送テストの他に様々な型を混合したテスト，burn-in テストなど 68 種類のテストを行った．

3.4.4 効果と課題

JBGEN 作成以前，手作業で AOCS タスクへの通信のプログラムコードを書いた際には数週間の作業期間を要していたが，JBGEN により作業時間は数分～数十分に短縮され，同時にプログラムミスの可能性も大幅に減少する効果を得た．JBGEN の課題としては以下が挙げられる．

- 現在，int，double 等の基本データ型，及びその配列の転送のみが可能である．オブジェクトの転送も行いたい．
- Java 側のプログラムと C 側のタスクは，お互いに次に呼び出されるメソッドを知っていて，期待される呼び出し順にメソッドを呼び出し，受け取らなければならない．エラー発生時に同期がとれなくなると，回復が極めて困難である．これを C 側ではイベントドリブンにし，Java 側から任意の順でメソッドを呼び出せるような構造にする必要がある．

JTRON 規格は JBlend 等，インストールベースの多い市販の組込み Java 処理系でも実装されており，JBGEN の改良を進めれば宇宙機開発以外の分野でも多くの開発プロジェクトに貢献するものと考えられる．

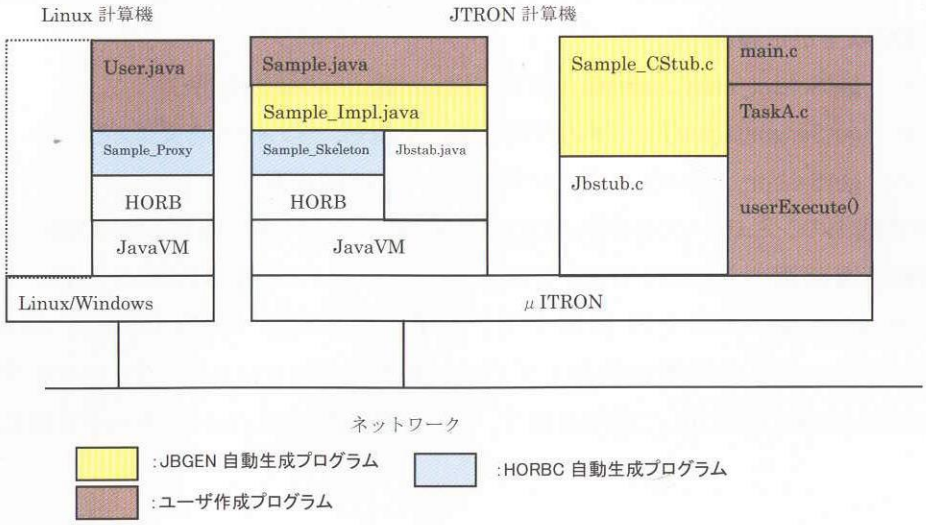


図 3-12 Java-μITRON ブリッジジェネレータ JBGEN の構成

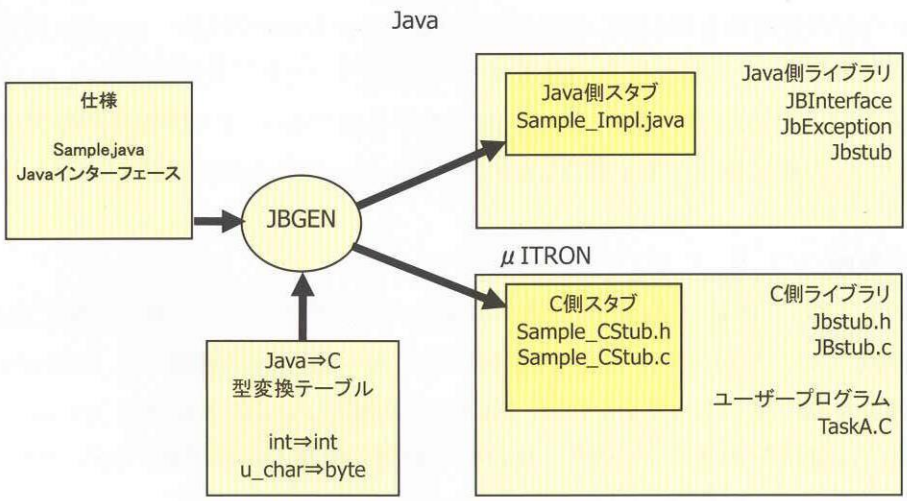


図 3-13 JBGEN の入力・出力ファイルの関係

4. 論理時間スケジューラの検討

従来の宇宙機シミュレーションは，各機能がシーケンシャルに作用するように実装することが一般的であった．例えば，一機の衛星の姿勢制御系，センサ，アクチュエータ，ダイナミクスの閉ループを考える際には，そのようなモデル化で十分な場合もある．しかし，通信，熱，電力といった複数の機能を複合的に模擬する場合や，複数衛星のシミュレーションを行う際には，図4-1に示すようにいくつかの機能が並行に動作すると考えるのが自然である．そこで，整合性を保ちながら複数の自律的なタスクを並行的に柔軟にスケジュール可能なシミュレーション・フレームワークが必要となる．

そこで，論理時間に従って進行する宇宙機シミュレーションはもとより，相互作用する複数タスクを実時間あるいは任意倍率で加速して実行させるためのスケジューラである分散シミュレーション・フレームワークを，近年広く用いられているマルチスレッド(multithread)技術を応用して開発している．表 4-1に，シングルスレッド(ステートマシン)によるシミュレーションとマルチスレッドによるシミュレーションの比較を示す．このスケジューラ上に，センサ，アクチュエータ，ダイナミクス等の自律オブジェクトをスレッドとして割り当て，さらに前節で述べたように分散シミュレーション技術を適用することにより，FSS だけでなく PILS に対しても同様な枠組みで実現可能である．

本スケジューラには，スレッド等の有用な機能を言語仕様に含み，実行形式が動作プラットフォームに依存しない Java 言語を採用している．Java の容易なメモリ管理機能や簡潔な言語仕様により，従来の言語を使用する場合に比べてメモリ・リークなどのバグが混入しにくい迅速なアプリケーション開発が期待できる．スケジューラで扱う自律的なモデルは，それぞれ別スレッドとして実装する．ただしシミュレーションは通常，論理時間に従って進行するため，実時間を扱う Java のスレッド API を直接使うことはできない．そこで，Java のスレッドクラスを論理時間に対応するように拡張し，図 4-2 に示すように各スレッドが協調動作しながら必要な論理時間にアクティブになるよう設計し，実装を行った．設計した論理時間スケジューラのクラス図を図 4-3 に示す．

例えば，HLA/RTI の適用性評価で用いた HTV/ISS 簡易シミュレータにこのスケジューラを適用することを考える．この場合は複数宇宙機のシミュレーションであり，HTV と ISS の姿勢，軌道ダイナミクスを独立に設計し，かつ並行に計算させることができる．また HTV の姿勢制御系の搭載ソフトを，前節で説明したように HORB，JavaVM および RTOS を搭載した別計算機で動作させ，さらに各計算機にスケジューラを載せて実時間同期させることにより，搭載ソフトの検証を行うことが可能となる．また搭載側の計算機として搭載 CPU をエミュレートするソフトウェアを用い，そのエミュレータの時刻を操作することで，論理時間加速した検証も可能と考えられる．

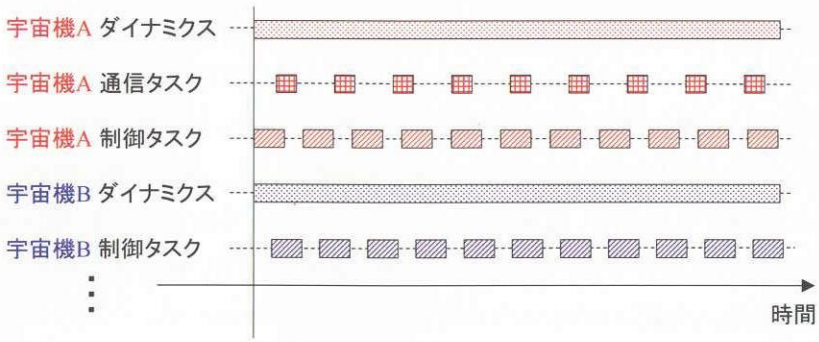


図 4-1 並行動作するタスク

表 4-1 シミュレータ実行方式の比較

	シングルスレッド（ステートマシン）	マルチスレッド
並行処理	×	○
時間概念の導入	実時間の扱い困難	実時間，論理時間の扱い容易
リアルタイムシステムとの親和性	×	◎（μITRON のタスクとは 1 対 1 対応）
実行速度	○	△（OS が介入，ただし，マルチ CPU，分散化により負荷分散容易）
アプリの複雑さ	×（システムにより条件分岐の数，組み合わせ数大）	○（状態遷移を自然に記述可能）
イベント処理の影響	×（イベント処理の間，全アクティビティが停止）	○（並行処理により影響小）
同期処理	必要なし	必要
デバッグ	△	×（難：論理的な検証が必要）
Java による実装	○	◎（Java 言語仕様に含まれる）

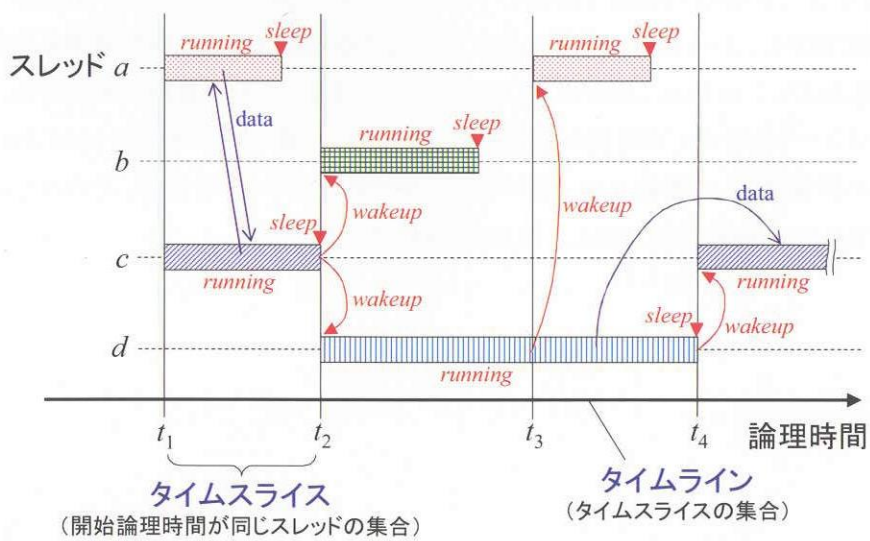


図 4-2 論理時間スケジューリング

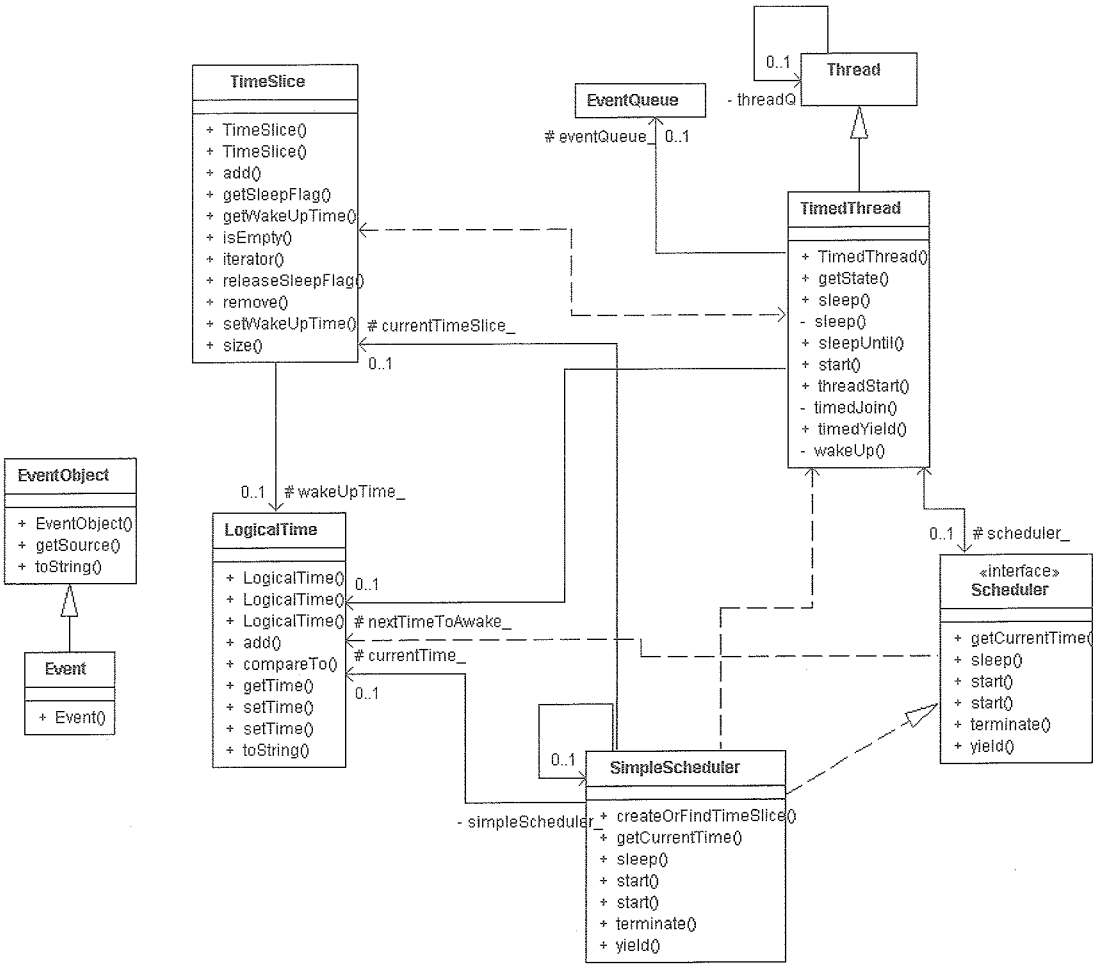


図 4-3　論理時間スケジューラのクラス図

5. まとめと今後の課題

これまで述べたように，本研究の成果は以下のようにまとめられる．

- 搭載計算機H/Wの違いを吸収するRTOS(μITRON)とJava，通信方式の違いを吸収する分散通信ミドルウェアHORBを利用して，RTOSと汎用OS上で連携するシミュレーション・フレームワークを提案した．
- シミュレーション形態(FSS, PILS)に依存せず，共通的に適用可能なI/Fの仕様を示した．
- MATLAB/Simulink, HORB，開発した JBGEN を利用し，可能な限りプログラムの自動生成を行うことにより，作業効率の向上，バグの低減を進めた．
- HORB のリアルタイム拡張，多言語対応を進めるとともに，宇宙機シミュレーション環境への適用性，有用性を確認した．
- 並行動作を柔軟に実現する論理時間スケジューラの設計と，論理時間でスレッドを切り替える機能の実装，動作確認を行った．

今後は，提案したシミュレーション・フレームワークの汎用性や信頼性を高めるため，実際の宇宙機の設計開発に適用しながら検討を進める必要がある．課題として，現状では通信プロトコルはTCP/IPのみを使用しているが，MIL-1553Bなど実際の宇宙機で利用されている通信方式への対応が求められる．これは基本的にHORBのプロトコル拡張機能を活用すれば現在のフレームワークがそのまま適用できると考えられる．ただし，そのためには通信デバイスや通信方式に依存するOBSのデバイス・ドライバを吸収する枠組が必要であり，本研究で試みたμITRONのメイルボックス機能を利用する方法もその一例である．

μITRONとJava間の通信プログラムを自動生成するために開発したJBGENに関しては，機能拡張を行っていくとともに，企業への技術移転を図る予定である．

また，論理時間スケジューラと積分機能等を統合すれば，宇宙機シミュレーション・フレームワーク全体の実行制御機能が実現できる．さらに，OBCに替えてプロセッサのS/Wエミュレータを用いることによって，OBSも論理時間加速して検証できるシステムの検討も重要であると考えている．

参考文献

[1] Satoshi Hirano, HORB: Distributed Execution of Java Programs, WWCA96, Lecture Notes in Computer Science, 127429-42, 1996.

[2] 上田裕子，高橋 孝，吉岡伸人，船原慎太郎，小堀壮彦，分散通信ミドルウェアHLA/RTIの宇宙機シミュレーションへの適用性評価，日本航空宇宙学会論文集，Vol.52， No.603， 2004，pp.160-166.

[3] Robert de Vries, EuroSim Simulation Framework, Data Systems In Aerospace 2003 (DASIA 2003), 2 June, 2003.

[4] Leo Timmermans, From simulation to operations - Developments in Test and Verification Equipment for Spacecraft, 1st ESA Space System Design, Verification & AIT Workshop, 13-14 June, 2002.

[5] Michael Schon, Marco Arcioni, Daniele Temperanza, and Kjeld Hjortnaes, Virtual Spacecraft Reference Facility, Data Systems In Aerospace 2004 (DASIA 2004), 1 July 2004.

[6] Manfred Bader, Andre Martin, and Reinhard Hendricks, Model Based Development Process - Opportunity for an Alternative AIV Strategy, 1st ESA Space System Design, Verification & AIT Workshop, 13-14 June, 2002.

[7] Frank de Bruin, Frédéric Deladerrière, and Fridtjof Siebert, A standard Java virtual machine for real-time embedded system, Data System In Aerospace 2003(DASIA 2003), 4 June, 2003.

[8] Greg Bollella, The Golden Gate Project: Real-Time Java in Space, Space Mission Challenge for IT 2003, 2003.

[9] http://www.corba.org/

[10] 坂村健監修, μITRON3.0 標準ハンドブック改訂新版, パーソナルメディア, 1998.

[11] Yukikazu Nakamoto, and Hiroaki Takada, JTRON: A Hybrid Architecture of the Java Runtime Environment and a Real-Time OS, Proc. of IEEE Workshop on Programming Languages for Real-Time Industrial Applications (PLRTIA), 1998, pp.85-93.

[12] http://www.personal-media.co.jp/

[13] 宇宙航空開発機構, 国際宇宙ステーション飛行継続に向けた NASA の対応について, http://www.jaxa.jp/press/2003/12/20031203_iss-01_j.html, 2003.

[14] 総務省, 情報システムに係る政府調達制度の見直しについて, http://www.soumu.go.jp/gyoukan/kanri/030329_1.html

[15] 平山、渡辺、組込みソフトウェア開発技術の動向、情報処理学会オブジェクト指向シンポジウム 2003.

[16] Object Management Group, CORBA2 Universal Networked Objects, 1995.

[17] Sun Microsystems, Remote Method Invocation Specification, 1997.

[18] Microsoft Corp., The Component Object Model Specification, 1995.

[19] W3C, SOAP Version 1.2, 2003.

[20] 平野,塚本, 分散 Java 実行のためのポータブルな ORB の構成法, SwoPP96 ,1996.

[21] 平野,組込み Java のためのオブジェクトシリアライザ生成法,オブジェクト指向シンポジウム 2001, 2001.

[22] HIRANO, Yasu, Igarashi, Performance Evaluation of Popular Distributed Object technologies for Java, Journal of Concurrency Practice and Experience, Vol10, pp927-940, 1998.

[23] Shapiro, M., Structure and Encapsulation in distributed Systems: The Proxy Principle, ICDCS, pp.198-205, 1986.

[24] The Real-Time for Java Experts Group (RTJEG), Real Time Specification for Java.

[25] Java-μITRON ブリッジジェネレータ jbgen1.0, JAXA 技術資料 GSB-04004

略語表

略 語	定 義
ACCL	Accelerometer
AIST	National Institute of Advanced Industrial Science and Technology
AOCS	Attitude and Orbit Control Subsystem
API	Application Program Interface
CCSDS	Consultative Committee for Space Data System
CORBA	Common Object Request Broker Architecture
DCOM	Distributed Component Object Model
DH	Data Handling Subsystem
ESA	European Space Agency
ESA	Earth Sensor Assembly
FSS	Full Software Simulation
GC	Garbage Collection
GPSR	Global Positioning System Receiver
GUI	Graphical User Interface
HILS	Hardware-in-the-Loop Simulation
HTV	H-IIA Transfer Vehicle
HORB	Hirano's Object Request Broker
HLA	High Level Architecture
H/W	Hardware
IDL	Interface Definition Language
I/F	Interface
IIOP	Internet Inter-ORB Protocol
IRU	Inertial Reference Unit
ISS	International Space Station
ITRON	Industrial TRON
IV&V	Independent Verification and Validation
JBGEN	Java-μITRON Bridge Generator
JNI	Java Native Interface
JPL	Jet Propulsion Laboratory
JTRON	Java Techology on TRON
MDVE	Model-based Development and Verification Environment
MPU	Micro Processing Unit

μITRON	Micro Industrial TRON
OBC	Onboard Computer
OBS	Onboard Software
OS	Operating System
PILS	Processor-in-the-Loop Simulation
RMI	Remote Method Invocation
RTI	Run-Time Infrastructure
RTOS	Real-Time Operating System
RTSJ	Real-Time Specification for Java
RTW	Real-Time Workshop
RVS	Rendezvous Sensor
SOAP	Simple Object Access Protocol
SVF	Software Validation Facility
S/W	Software
TRON	The Real-Time Operating System Nucleus
TTC	Telemetry, Tracking & Command Subsystem
TVE	Test and Verification Equipment
VM	Virtual Machine
VS-RF	Virtual Satellite Reference Facility
WSDL	Web Services Description Language

宇宙航空研究開発機構研究開発報告　JAXA−RR−04−017

発　行　日　　2005年1月31日
編集・発行　　独立行政法人 宇宙航空研究開発機構
〒182−8522
東京都調布市深大寺東町七丁目44番地1
TEL 0422−40−3000（代表）
印　刷　所　　株式会社 ビー・シー・シー
東京都港区浜松町2−4−1

＜本資料に関するお問い合わせ先＞
独立行政法人 宇宙航空研究開発機構 情報化推進部 宇宙航空文献資料センター

宇宙航空研究開発機構
Japan Aerospace Exploration Agency