ロボットシミュレータのためのオープンなフレームワークの実装

An Open Robot Simulation Environment

石村 俊幸 [1)]　小田 謙太郎 [2)]　加藤 健士 [3)]
Toshiyuki ISHIMURA　Kentaro ODA　Takeshi KATO

1) 九州工業大学大学院 情報科学専攻（〒820-8502 福岡県飯塚市大字川津 680-4
E-mail: isshi@mickey.ai.kyutech.ac.jp ）

2) 九州工業大学大学院 情報科学専攻（〒820-8502 福岡県飯塚市大字川津 680-4
E-mail: ken@mickey.ai.kyutech.ac.jp ）

3) 九州工業大学大学院 情報科学専攻（〒820-8502 福岡県飯塚市大字川津 680-4
E-mail: kato@mickey.ai.kyutech.ac.jp ）

ABSTRACT. Nowadays various kinds of robots are publicly available such as AIBO, ASIMO and so on. However, the development of robots is still difficult because of their complexity, continual changes of environments, limitation of resources and so on. To solve this problem, robot developers often use the simulator that allows to program and test robots' program effectively in the ideal environment where specified conditions can easily be reproduced. It is still difficult to realize the simulator regardless of its usefulness, because the cost of simulator implementation seems the unexpected cost in the development of robots. To overcome this problem, it is need to realize the open robot simulation environment in which any kind of robots can be simulated. This paper focuses on vision-based robot simulation environment and describes a method to construct it. Finally, we implemented a simulator for Robocup Sony 4-Legged League by this method.

1　はじめに

現在、エンターテイメントロボットを始めとする多種のロボットが開発され、実際に市場に出回っています。また求められる機能も複雑化しており、ロボット開発の効率化・高速化が望まれています。このような問題を解決するための手段の一つとしてシミュレータの導入が考えられます。シミュレータを導入することで実機を使用せずにテストが行え、また理想的な環境下でのロボットの振る舞いの確認などが可能になり、ロボット開発をより円滑に行うことができます。このロボットシミュレータには様々なものが考えられます。任意の行動にセンサ情報を対応させたただの表もシミュレータと言えるかもしれません。他方では、行動としてエフェクタに値を与え、センサ入力としてカメラ画像や音声を扱うロボットのシミュレータはより複雑になります。また、仮想的にロボットや環境を扱えるという点から、事物を抽象的に扱うシミュレータも考えられます。例えば、RoboCup の Simulation league のように、具体的なロボットの構造や環境との相互作用を厳密に考慮せず、プランニングを重視するロボットのためのシミュレータなどです。つまり、ロボットシミュレータは、目的やロボットに応じて幾つかのレベルに分けられると考えられます。このように多岐にわたるロボットに対するシミュレータは、それぞれのロボットの開発者によってそのロボット専用のものが実装されます。これは、大変なコストが必要になります。

さらに、シミュレータ開発のためにはロボット開発以外の技術が必要になります。また、本来のロボットの開発とは別にシミュレータを開発するためのコストが必要となる点から困難であると言えます。このためその利便性の関わらずシミュレータの導入は容易であるとは言えません。

一方で、シミュレータにおいて、カメラ画像のようにロボットにとって一般的なセンサ情報・エフェクタ情報の取得は、実機の構造に多少影響を受けるもののほとんど同じ枠組みであると考えられます。例えば、カメラ画像は、カメラの位置や向き、カメラ固有のパラメータなどが異なるだけで、シミュレータ内部での画像生成や取得の部分は同じプログラムで、複数のシミュレータに対応できます。他にも、ロボットと環境との衝突判定、環境中のオブジェクトの形状情報を定義する機能なども同じことが言えます。

本提案では、このようにシミュレータを扱うためのインタフェースや、仮想環境やロボットの情報の登録や取得のためのオープンなインタフェース、基本機能を持ったライブラリを提供するプラットフォームを実現することを目標にします。これにより、シミュレータ作製者は煩わしい部分を極力実装することなく、シミュレータを実現できることが期待できます。つまり、誰でも簡単に素早くロボットシミュレータの実装を行えるフレームワークの実現が目的です。

2　目的

シミュレータは、ロボットの機械的な挙動をシミュレートするものと、自律型ロボットのためのものに分類できると考えています。前者は、ロボットの機械設計に用いられロボットの形状モデルデータを重視したものや、ロボットアームのための制御プログラム生成用のものがあります。一方、後者ではロボットは自身が取得した画像（ビジョン）を元に自律的に行動します。このようなロボットが活動する環境は動的に変わることが多く、問題が発生した場合もバグなのかノイズなのか判別しにくく再現性が低いという問題点を抱えています。本提案では、後者のビジョンベースロボットの開発を助けることを目標としています。そこ

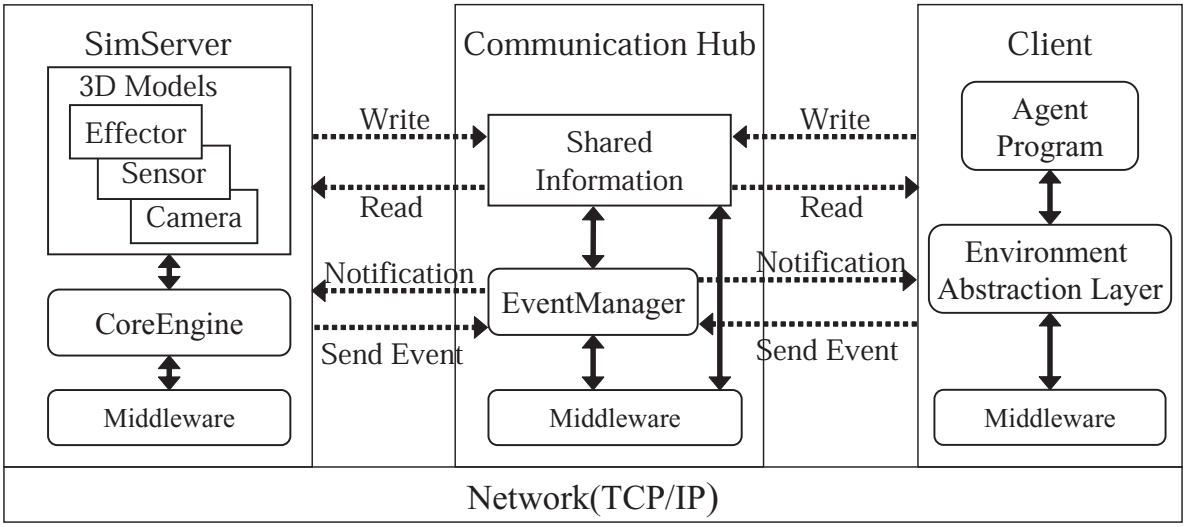

図 1: システムアーキテクチャ

で、以下にあげる項目について考察し、実現を目指します。

- オープンさ： ロボット開発者、つまりシミュレーション環境が使用者に対して、内部まで良く見え、細部まで操作できること
- シミュレーションの再現性を確保
- ロボットプログラムの変更を最小にする
- 使いやすいテスト・モニタツールの実現

一つ目の項目についは、本提案において最も重視した項目です。高いオープン性は複数種類のロボットをシミュレーション環境に参加させることができ、実際の環境に合わせてシミュレーション環境をカスタマイズし易くします。また、現実では再現させにくい環境（例えば、劣悪な照明条件や故障したロボット）のを再現することも容易になります。

二つ目の項目はロボット開発において重要となります。実機（現実のロボット）は常に環境の変化（照明条件など）の悪影響を受けます。このような場合、ロボットの戦略プログラムの予期しない挙動が、環境の変化によるものなのか、プログラムのバグによるものなのかの特定が著しく困難になります。このような予期しない挙動を特定するには、理想的で且つ同等の環境、つまり、再現性のある環境でのテストが必要となります。実際のプログラム開発では、コーディングとテストを繰り返しながら、開発を進めていきます。ロボット開発でも同じで、戦略プログラムのコーディングとテストを繰り返しながら、ロボットの戦略プログラムを開発していきます。しかし、ロボット開発においては組み込みプログラミングであることが多いため、テストにかかるオーバヘッドが大きいという問題があります。例えば、プログラムのコーディング後テストを行うために、そのプログラムをロボットに移すのが困難な場合もありますし、ロボットや周りの環境をテストしたい状態にすることに時間を費やす場合もあります。このような理由から、再現性の確保・実現はシミュレーション環境にとっては重要な要素の一つであると考えます。

三つ目の項目は、シミュレーション環境の構築のコストを下げることが目的です。最初にも述べたようにシミュレータは、その利便性にも関わらずコスト面や技術面で導入が困難な場合があります。このような問題を回避する上でも、実機用のロボット戦略プログラムをシミュレータ用に移行するコストを抑える必要があると考えます。

最後の項目は、ロボット開発者を助けることが目的です。実際にロボット戦略プログラムをシミュレータ上で稼動させることができても、ロボットの挙動やシミュレーションの内部情報を簡単に閲覧できなければ、意味がありません。そこで、シミュレーションされているロボットを含めた環境全体の情報を簡単に閲覧でき、さらには編集できる枠組みを提供する必要があると考えます。

3 開発内容

一般的なシミュレータはシミュレーションを行うモジュールとロボット戦略プログラムを同じ環境で扱うアーキテクチャを採っているものが殆どです。このアーキテクチャでは本提案の目標を十分に達成できないので、我々は図 1 に示すようなアーキテクチャを採用しました。ここでは、クライアント/サーバモデルを採っています。そして、サーバとして、二つの *SimServer* と *Cummunication Hub* というサーバを導入しています。

SimServer　このサーバは全ての仮想シミュレーション環境のオブジェクトを管理し、仮想カメラのイメージ生成や、ロボットのエフェクタのシミュレーションを行います。

Cummunication Hub　*SimServer* と各種クライアントとの通信を管理します。また、通信内容を常に監視しています。

異種環境で実装されるロボット戦略プログラムを想定している本提案では、その異種性を吸収するため、シミュレーションを実現する部分と、戦略プログラムを分離する必要があります。このことを実現するために、クライアント/サーバモデルを採り、二つのサーバを導入しています。また、複数のロボット戦略プログラムを同じシミュレーション環境で柔軟に扱えるようにするためにも、このモデルを採用しました。

もし、実機がシミュレーション環境と通信でき、自身の情報を送信することができるならば、仮想ロボットと実機が同じシミュレーション環境で動作するといったことも可能です。また、*Cummunication Hub* に接続されるのはロボットプログラムだけに限定されません。シミュレーション環境中の様々な情報を取り出せるため、例えば、独自のモニタリングツールの作成も比較的簡単に行えると考えます。このように、さまざまなロボット戦略プログラムやクライアントを柔軟に扱えるようにするために、このアーキテクチャを採っています。

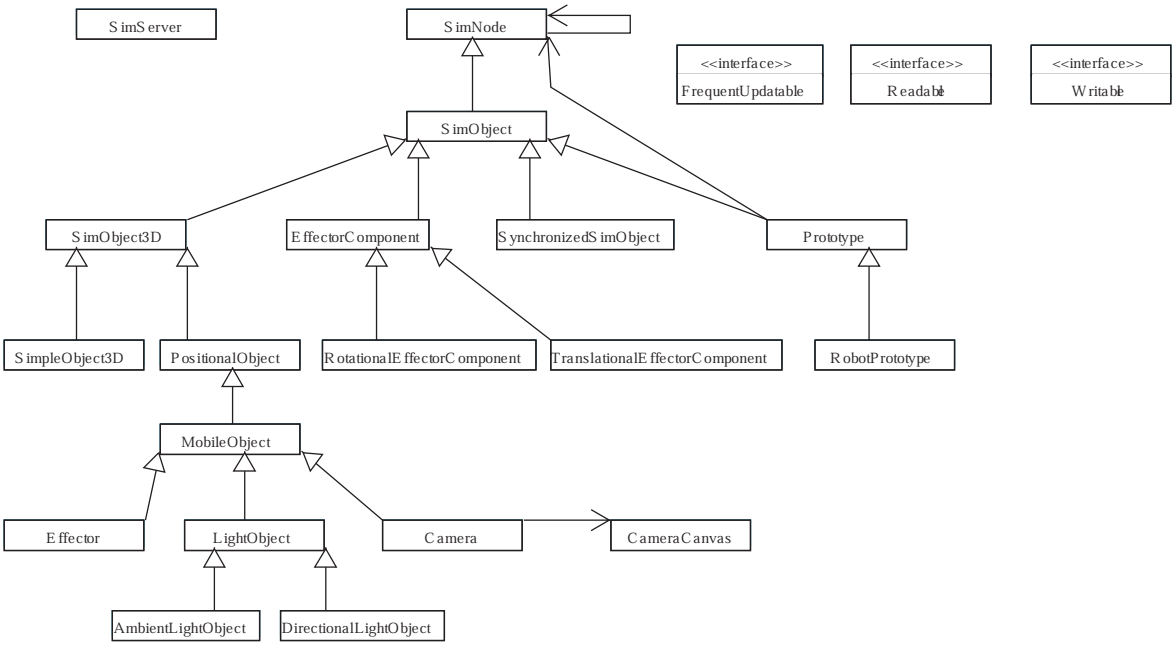

図 2: クラス階層

■オブジェクトモデル　*SimServer* によって管理されるオブジェクトモデルを図 2 に示します。SimObject3D の子クラスは、実際に 3 次元 CG のオブジェクトと関連付けられるオブジェクトを表現します。例えば、ロボット自身や、カメラ、エフェクタなどがあります。つまり、ロボットを構成する代表的なオブジェクトは、MobileObject、Camera、Effector となります。Effector は EffectorComponent を子ノードとして持ちます。この EffectorComponent は、一つの関節に複数の自由度がある場合、個々の自由度を表現します。例えば、人間の首の場合、頷く方向と横に振る方向の 2 つの自由度があるといえます。ここで、首を Effector で表現し、各方向を RotationalEffectorComponent で表現することで、柔軟に関節をモデル化することができます。同期が必要なロボット、例えば、何らかの動くカメラ（頭に搭載されるカメラ）を装備しているロボットは、エフェクタのセンサ値と取得される仮想カメラ画像が一致している必要があります。つまり、首を振った場合は、その角度の仮想カメラ画像を必要とするロボットです。このような場合は、ロボットを SynchronizedSimObject で表現することで、ロボット戦略プログラムから送られた要求と *SimServer* からの結果が同期される通信を提供することができます。逆に特に指定しない場合は、非同期で通信が行われ、同期性は保障されません。本提案が目標にするロボットシミュレーションでは、複数のロボットを扱うことを前提にしています。さらに、これらのロボットが動的に追加・削除できる機能を提供します。これを実現するために、ロボットの情報（SimObject 群と 3 次元 CG のオブジェクト群）を格納する Prototype があります。これは、実行時にはシミュレーション環境中には現れませんが、特定のイベントを受けるとその Prototype を元にロボットを生成し、シミュレーション環境に参加させます。

■*SimServer*　*SimServer* は、図 2 に示されているオブジェクトからを基にした仮想シミュレーション空間を構成する SimObject 群と CoreEngine に分かれます。さらに CoreEngine は大きく以下の構成に分かれます。

Visualizer　シミュレーション環境の可視化を行います

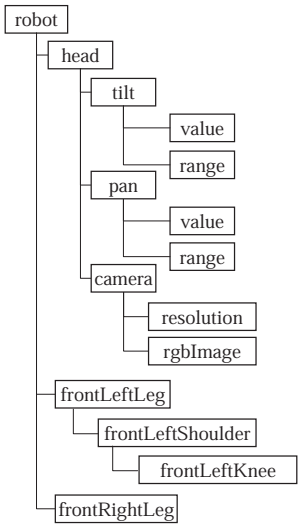

図 3: 名前空間の例

ScriptEvaluator　後で述べるスクリプトを評価し実行します。

オブジェクト管理　Prototype の管理や SimObject の追加や削除を管理します。また、同時に 3 次元 CG オブジェクトの管理も行います。

プラグイン管理　プラグイン・モジュールの有効・無効、追加・削除を管理します。

■木構造に基づく名前空間　ロボットは、時として複雑な構造を持つため、エフェクタやカメラなどを普通の名前空間で管理すると、利便性を著しく損なう恐れがあります。そこで、本提案のシミュレーション環境中の各 SimObject は名前を持っており木構造に基づく階層構造で *SimServer* に管理されます。例えば、各オブジェクトには `/Robo0/robot/head/camera/image` のようなパス名でアクセスすることができます。これにより、名前の意味と場所を確認しながら、アクセスすることができ、最上位の

ノードを削除することで簡単に削除が行えます。このように全てのオブジェクトにわかり易く共通のアクセス方法を提供することで、ロボット開発者の負担を軽減します。図3にオブジェクトの階層構造の一例を示します。

一方、全てのクライアントは、システム情報を含む全てのオブジェクトにアクセスすることが可能です。このとこは、現実環境では得ることが困難な情報を必要とするテストも行うことが可能です。例えば、カメラ画像を元に自己位置同定を行うようなロボットを考えます。シミュレーション環境においては、認識結果の位置と、理想的な自身の位置とを比較でき、且つ任意のオブジェクトとの距離を画像認識で計測するような場合でも、理想的な距離との比較を随時行うことができ、自己位置同定の精度を確認することが容易に行えます。

■*Cummunication Hub* シミュレータ内部のフレームワーク/プラグインという Java 言語に密着したオープン性を提供するだけではなく、分散異種混合環境におけるオープン性を提供します。また、クライアントの接続・切断からシステム全体の停止を防止することを目標にしています。以下のモデルを元に実装しています。

共有情報モデル 木構造による外部表現、その read/write 操作による値の書き込み、読み込みができる。つまり、先に述べた木構造に基づく名前空間によって *Cummunication Hub* にアクセスできます。

イベントモデル イベント送出、受信、イベントには文字列ラベルをつける。文字列ラベルの正規表現によるマッチングで、受信イベントの選別を行う。これにより、比較的簡単に同期通信を導入できます。

また、再現性の確保やデバッグを目的として、*Cummunication Hub* は自身を介してやりとりされる情報を永続化し、復元する機能を持たせます。永続化された情報はファイルの格納し、いつでも参照できるようにすることにより、シミュレーション環境中の情報を用意に扱えるようにします。

■通信ライブラリ 様々なロボット戦略プログラムを想定している本提案では、そこで交わされる情報の全てを予測することは不可能です。また、シミュレータと戦略プログラムの間では、画像を含む多くの情報が頻繁に交換されます。つまり、複雑なプロトコルは、ロボット開発者を混乱させるだけでなく、パフォーマンスの面でも問題が起きることが想定されます。このような背景から、非常にシンプルなバイナリ通信プロトコル（名称：Ethereal）を実装しました。これは、プリミティブ型（byte, int, double, boolean）と文字列型、これらの配列のみを扱うプロトコルです。このプロトコルを扱うライブラリとして、Java 版と C++版を用意します。

C++ 版の通信ライブラリは、 Java 版のライブラリと同じように使うことが可能なインタフェースを提供することを目的としています。 これによりロボットプログラムの開発言語が C++ から Java へ変更されるといった変更があった場合に、 移行コストを下げることが可能になります。また C++ における実装ではどうしても実行環境に依存するコードが必要となります。 今回の実装では Linux や FreeBSD といった Unix 系の OS を対象としていますが、 それ以外の環境、例えば Windows といった環境へ移植する際のコストを下げるために、 環境に依存するコードをできるだけ分離することも目指しました。 そこで C++ 版の通信ライブラリは以下の三つの部品から構成されます。

- 環境に依存しない、 メモリリーク対策などの機構を提供するためのクラスや関数群
- 環境に依存するコードを含むクラスや関数。(例:socket を用いた TCP/IP による通信など)
- *Cummunication Hub* と実際に通信するためのクラス

このような構成にすることで、 *Cummunication Hub* との通信を行うクラスには手を加えずに socket など環境に依存した部分のみを差し替えることで、 移植することが可能となります。

```
(let
  (set /Ball/loc/x
    (plus /ribo0/loc/x
      (/ (dist /robo0 /robo1) 2)
    )
  )
  (set /Ball/loc/y
    (plus /ribo0/loc/y
      (/ (dist /robo0 /robo1) 2)
    )
  )
)
```

図 4: スクリプト言語の例

■プラグイン・モジュール 必要とされる機能を全て実装しておくことができれば、理想的なフレームワークと呼べるかもしれませんが、予めそれらを全て予測することは非常に困難です。そこれ本提案ではプラグイン・モジュールという形でユーザが機能をある程度拡張できる枠組みを提供します。現在以下のようなプラグインのテンプレートが存在します。

画像変換用プラグイン 画像をユーザが操作するためのプラグインです。*SimServer* で生成された画像を実際に、戦略プログラムに送信する前に編集することが可能です。例えば、故意にノイズをのせるプラグインなどが考えられます。我々は、実際にこのようなプラグインを幾つか作成し、戦略プログラムのバグ取りに使用しています。

エフェクタプラグイン エフェクタに値が適用される前に、その値に対してフィルタをかけることを目的としてプラグインです。例えば、減速を実現するプラグインなどが考えられます。実際に、後で述べるシミュレータでの AIBO にはこのプラグインを搭載して、各エフェクタの減速を擬似的に再現しています。

制約プラグイン ありえない値を防ぐためのプラグインです。例えば、位置制約プラグインなどがあります。本来、コンピュータグラフィクスの衝突判定は非常にコストの高い処理です。しかし、移動範囲を限定するなどの処理は、制約を予めユーザが与えておくことにより、少ないコストで実現できます。

イベントプラグイン *Cummunication Hub* から送信されるイベントに対して起動するプラグインです。クライアントから送信されるイベントに対して、特別な処理が必要な場合などに使用します。

これらプラグイン・モジュールは実行時に追加・削除、有効・無効が可能で、仮想ロボットの挙動をカスタマイズするだけでなく、デバッグにも使用できるような枠組みになっています。

■OpenSim スクリプト言語 ロボットを含めるとシミュレーション環境中のオブジェクトは相当数存在し、種類も様々です。これら各オブジェクトを操作するインタフェースの実現に、一般的な GUI を用いるのは、問題があります。よく設計された GUI はユーザを大いに助けるでしょ

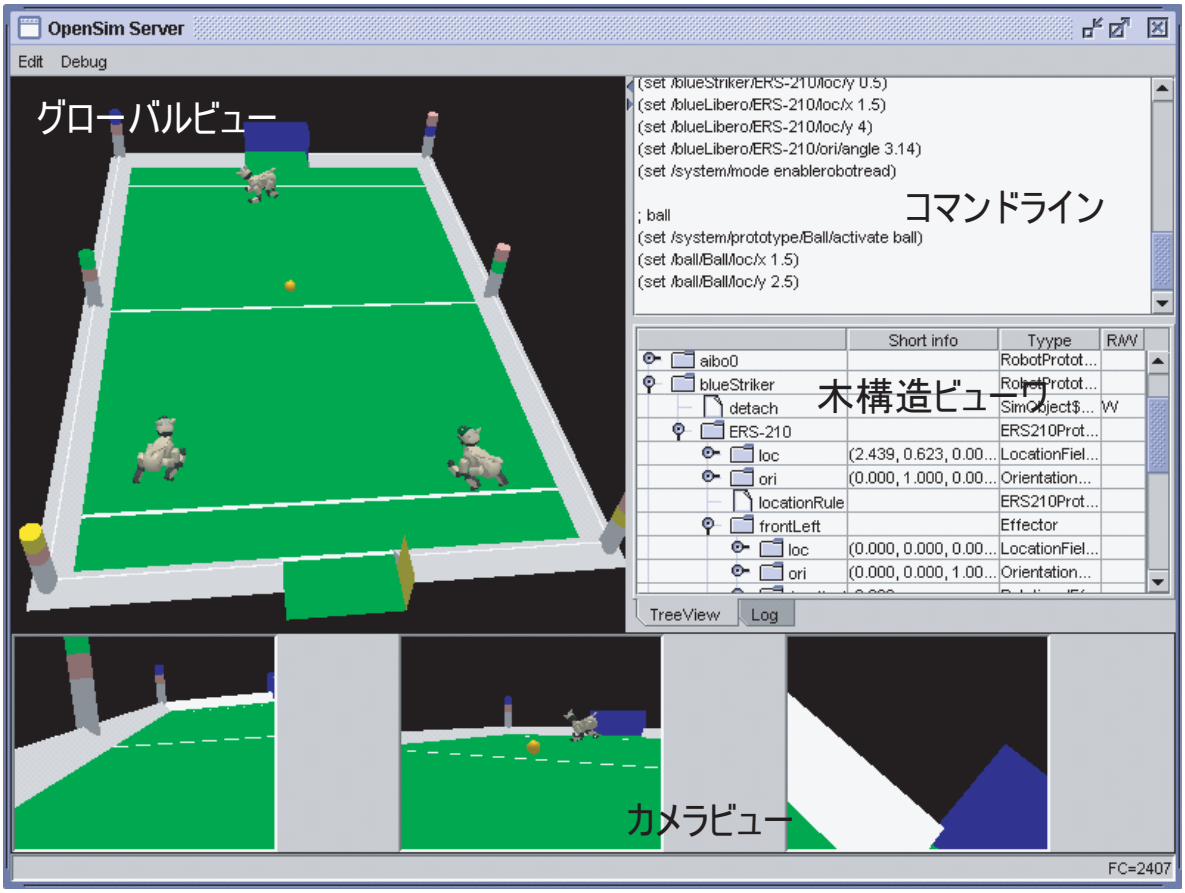

図 5: *SimServer* の概観

うが、様々な種類のオブジェクトに対しオープンで柔軟性のある GUI を提供するのは困難です。そこで、本提案では簡単なスクリプト言語を提供することによってこの問題を解決します。言語は GUI に比べて多少習得に時間が必要ですが、その柔軟性はこの欠点を補って有り余るものがあります。

本提案で実装した言語（以降、OpenSim スクリプト）は図 4 に示されるように S 式を基にしています。言語レベルの基本的なコマンドを始め、*SimServer* や *Cummunication Hub* に依存するようなコマンドを実現しています。図 4 は、`robo0` と `robo1` の間にボールを移動させる例です。

4 実装

SimServer と *Cummunication Hub* サーバは Java 言語を、3 次元 CG の部分は Java3D を使用して実装しました。Java 言語はインタプリタ言語であるために、その実行速度の面でやや劣る面があるものの、複数のプラットフォームに対応可能であり、オブジェクト指向やガーベッジコレクション機能などにより、開発速度の面で大きな利点があると考え、Java 言語を選択しました。実際に本提案で作成したサーバは Windows 上と、Linux 上で動作を確認しています。通信ライブラリの方は、先に述べたように Java と C++の二つの言語用のものを実装しています。

SimServer は、図 5 に示すような GUI を持っています。グローバルビュー、コマンドライン、木構造ビューワ、カメラビュー、ログコンソールの五つのコンポーネントから構成されています。

グローバルビュー シミュレーション環境をユーザの視点で見ることができます。マウス操作で簡単に視点を変更できます。また、キャプチャ機能も有しており、その画像を PNG 形式で保存することが可能です。

コマンドライン 先に述べた OpenSim スクリプトを実行するためのコンポーネントです。各種コマンドを実行でき、結果を得ることができます。

木構造ビューワ 環境中の全てのオブジェクトを木構造に沿って、閲覧することができます。情報の更新はリアルタイムで行われます。

カメラビュー 現在、シミュレーション環境に存在するカメラの画像を表示します。

ログコンソール システムの詳細なログを閲覧することができます。また、オプションを調整することにより、情報の粒度を変更することができます。

5 評価

RoboCup 用シミュレータの実現 本フレームワークを利用して、ような RoboCup Sony 4-Legged League のシミュレータを実現しました。このシミュレータは Java の主要技術の一つである Java Web Start を使用して、公開されています。また、戦略プログラムを含むクライアントについてもソースコードが公開されており、GCC にてコンパイルすれば、すぐにでもシミュレーションの様子を確認することが可能です。

オブジェクトモデル 17 自由度を持つ AIBO を先に述べた SimObject を使用すると、わずか 200 行足らずでモデル化できました。このコード量は決し多くなく、このオブジェクトモデルがビジョンベースロボットのモデル化に有効であると考えています。

OpenSim スクリプト　本提案を実現する上で、我々開発者一同も実行時にシミュレーション環境の情報に簡単にアクセスする必要に迫られました。そこで OpenSim スクリプト言語を実装したのですが、機能を少し拡張することにより、シミュレーション環境の初期化や、実行時のカスタマイズにも使用できるものになりました。実際に後で述べる RoboCup 用のシミュレータ上でのデバッグなどにも活用されていることからその有用性は高いと言えると思います。

ロボット間通信　*Cummunication Hub* の状態を保持する機能を使用して、ロボット間通信に適用できないかと考え、先に述べた RoboCup 用のシミュレータを上でこの評価を行いました。AIBO は無線 LAN カードを装備しており、これを用いて通信を行うことができます。実機の戦略プログラムもロボット間で通信行い、環境中の情報を共有しています。これにより、例えば、ボールを見つけていないロボットでも、ボールを見つけているロボットからの情報で即座にボールの位置を把握できるようになっています。

このボール情報を *Cummunication Hub* に仲介させる方法で行ったところ、簡単に通信をシミュレートできることが確認できました。具体的には、各ロボットが自身の名前と自身が持つ情報を *Cummunication Hub* に書き込むコードと、イベント通知を受けてその情報を元に読み込むコードを戦略プログラムに追加しただけです。*SimServer* や *Cummunication Hub* にはこの機能の実現のために一切手を加えていません。このことから、*Cummunication Hub* のオープン性と柔軟性、*SimServer* と *Cummunication Hub* の独立性の高さを実現できていることが分かります。

***Cummunication Hub* による環境の XML ファイルへの永続化**　*Cummunication Hub* の環境中の情報を XML へ保存したり、復元したりする機能を使用して、シミュレーション環境の再現性を確保する評価を行いました。保存や復元のコマンドは OpenSim スクリプトにより実装されており、簡単なスクリプトで行えます。

実際に保存と復元を行ったところ、保存された状態に復帰できることを確認しました。また、XML 形式で保存されているので、ヒューマンリーダブルですし、内容を解析し可視化するツールの開発も比較的容易であると考えています。また、定期的に環境中の情報を保存したり、ある一定期間連続的に保存したりする機能を使用すれば、デバッグにも大変有効であると考えています。

6　まとめ

本提案は、我々開発者一同が参加した Robocup がきっかけになっています。そこでシミュレータを作成していたのですが、最初は開発者とシミュレータ使用者とのギャップや、あまりの要求の多さに苦労しました。そこでの、反省点と今後の発展を考えこの提案を実現に移すことにしました。

当初の目標通りに、実現できなかった部分もいくつかありますが、副作用的に実現された機能もありました。実際に、ここで作成した成果は我々の Robocup チームで開発に使用されていて、その成果を着実にあげています。

今後は、Robocup に参戦した我々のチームのホームページである　`http://www.asura.ac`
にて、最新版のソースコードやドキュメントを、適切なオープンソースライセンスの元で本提案の成果を公開する予定です。

また、現状では、ユーザインタフェースの機能は必ずしも十分とは言えません。また、*SimServer* の機能もさらに拡張する余地が残されていると考えています。その他にも以下のような項目について改良・実現を図っていきます。

- 他環境での評価
- ユーザインタフェースの精錬
- 衝突検知の実現
- 汎用的なモニタリングツールの開発

最後になりましたが、我々のチームに関心を持って助言して頂いた竹内郁雄教授に感謝いたします。