自律分散システム構築用プラットフォームと 機能ツールの開発*

林 朗 弘** 佐 竹 利 文*** 鈴 木 裕**

Development of Platform and Functional Tool Kit for Constructing/Executing
a Decentralized Autonomous System

Akihiro HAYASHI, Toshifumi SATAKE and Hiroshi SUZUKI

This paper presents a design of the software platform and framework intended to develop and execute a decentralized autonomous system. The platform has the featured functions that sub-system can be transported on a distributed hardware environment, and functions of sub-system can be remade by joining different two sub-systems together and by separating a sub-system. By using the platform, system designer can easily construct the application that consists of sub-systems behaved as taking in functions of other sub-system and moving on network, if necessary. Furthermore, these functions can be performed by a sub-system remote from the sub-system to be transported, joined or separated. Improving capabilities of system implementation and maintenance can be expected by this remote execution. In this paper, the platform and framework developed in this study are described, then result of an example system implemented into the developed platform is reported for discussing usage and effectiveness of the platform.

Key words: decentralized autonomous system, programming environment, sub-system transportation, sub-system connection and separation, computer network

1. は じ め に

大規模で複雑なシステムを構築するための方法論として自律分散概念が注目されている。自律分散概念に基づく自律分散システムは、全体を管理するコーディネータをもたず、その構成要素である複数のサブシステムが、相互作用しながら全体としての秩序を形成するように動作するシステムである。自律分散システムでは、サブシステムが独立に動作するため、信頼性や耐故障性の上で有利である。また、自律的に秩序を形成するので、仕様や目的の変更に柔軟に対応できることが期待できる。このような自律分散システムを構成するには、サブシステムを協調させる方法と、自律分散システムを実装／運用できるプラットフォーム（システムの実装環境）やフレームワーク（システムを構築するためのツール群とそれを利用するための規則）の開発が必要である。前者についてはこれまで幾つかの手法^{1)～3)}が議論されている。しかし後者については、アプリケーションごとに個別の対応が取られており、広く利用できる技術として十分に整理されていない。

自律分散システムとして構築されたアプリケーションでは、内部的な作用や外部操作によって、その機能や構造を動的に変更できることが重要である。特に、サブシステムの追加や削除といったサブシステムの単位の構造変更だけでなく、相互作用といったサブシステムの協調機能に関わる機能の変更も動的に行えることが必要となる。また、サブシステム間の協調動作は通信などを介した情報交換が基本となることから、構造の動的な変化に対して透過な通信機能も必要である。しかし、これらをアプリケーションの問題として処理しようとすると、サブシステムの記述が複雑なものになる。このような機能は、自律分散システムを実装／実行するプラットフォームを設け、その中に内在させる方が望ましい。また、コンピュータのダウンサイジングとネットワーク化により、システムの

実装／運用環境は分散ハードウェアへ確実に移行しつつある。自律分散システムも分散ハードウェアを実装／実行環境としなければならない。利用可能な技術として、分散オブジェクト指向言語⁴⁾やクライアント／サーバ形システムの通信の標準化^{5) 6)}などが進められている。しかし、サブシステムの独立性や機能の動的な変更に対して自律分散システムの利点を活用するには必ずしも十分ではない。

本研究では、分散ハードウェアが提供する計算環境を対象に、自律分散システムとしてアプリケーションを構築する方法について検討し、その実装／運用環境となるプラットフォームとフレームワークを開発してきた。開発したプラットフォームは、サブシステムが分散ハードウェア環境の中を移動する移送、複数のサブシステムを結合してひとつのサブシステムとする結合、サブシステムを2つのサブシステムに分ける分離機能を持ち、アプリケーション実行時にサブシステムの機能を追加、置き換え、削除できる特徴をもつ。アプリケーションは、メッセージ送信と、それを受け取ったサブシステムがトランザクションを実行するというイベント駆動スキーマにしたがって動作する。サブシステム間のメッセージの送受信や情報の通信はブローカを介して行い、ブローカがサブシステムの実装位置に関する情報を動的に管理することにより、サブシステムに対し実装位置透過な通信機能を提供する。以下では、自律分散システムとして構築されたシステムをアプリケーション、その構成要素をサブシステムと呼び、開発したプラットフォームとフレームワークの構成と機能について報告する。

2. プラットフォームの基本機能

2.1 利用可能な技術と問題点

分散ハードウェア環境は、異なるOSにより管理された異機種種のコンピュータからなるのが一般的である。アプリケーションは、OSの多様性を認めながらそのような環境を意識せずに構築することが必要となる。多様なOSの上でアプリケーションの開発および実行環境を提供するプラットフォームとして、

* 原稿受付 平成9年2月17日

** 正 会 員 九州工業大学情報工学部（飯塚市川津680-4）

*** 正 会 員 沼津工業高等専門学校（沼津市大岡3600）

OSFのDCE⁵⁾やOMGのCORBA⁶⁾に代表されるようなミドルウェアがある。これらは、トランザクションの呼出し方法と通信の仕様を標準化することにより、分散して存在する情報や機能などを異なるアプリケーションから相互に利用できる環境を提供する。しかし、これらは、クライアント/サーバシステムのようなタスクを共有利用する形式のアプリケーションを想定しており、自律分散システムのようなピアトゥピアなデータインタラクションが頻繁に行われるシステムへの配慮が不十分である。例えば、CORBAはクライアントをサーバとしてあるいはサーバをクライアントとして構成し直すことができるが、設定を変更するという再実装作業が必要となる。また、限られたハードウェア環境で複数の異なるアプリケーションを実行する場合や、特定のサブシステムにネットワークを介したデータインタラクションが集中したりサブシステムの仕様変更が頻繁にある場合などに対して、サブシステムの実装方法と運用管理の形式やメンテナンスの方法を提供していない。

自律分散システムでは、システム構成や機能を動的に変更できることが要求される。また、サブシステム間の相互作用によって全体としての機能が発現することになる。そのため、ネットワークトポロジやアプリケーションやサブシステムの仕様変更などシステムの動的な変更に対応できるスケラビリティと開放性や、アプリケーション別にサブシステムを管理し必要に応じてアプリケーションにまたがる情報操作を隠蔽する機構が必要となる。

アプリケーションのポータビリティとインタオペラビリティを改善するツールとしてオブジェクト指向の概念に基づく開放型分散システムOZ++⁴⁾やHORB⁷⁾がある。OZ++は、オブジェクトとして表現した情報のクラスを分散システム全体で管理し、オブジェクトをネットワーク内で転送可能とすることにより、オブジェクト指向の利点であるクラスの利用や変更あるいは拡張を実現している。システムは、専用のオブジェクト指向言語で記述し、言語処理系が分散したオブジェクトを稼働させることにより機種に依存しない実行環境を実現している。しかし、耐故障性や信頼性という面から見れば、サブシステムは論理的にも動作単位としても独立である必要がある。OZ++では動作単位としてオブジェクトの独立性が考慮されていない。一方、HORBは、Java環境をベースに分散ハードウェア環境にまたがるオブジェクト間の呼出しや通信をプログラムできるCORBAと同等の機能を付加したツールと見ることができる。Javaをベースとしていることから、HORB言語の仕様で書いたアプリケーションは、OSの異なるコンピュータからなる分散ハードウェア環境で動作させることができる。しかし、このような専用の言語の使用は、専用の開発環境と言語処理系が実装されたハードウェア環境が必要となる。CORBAなどは言語に依存しないプラットフォームを提供しているが、ソフトウェア資産の再利用という面からも専用言語の仕様に拘束されない形態の方が望ましい。また、HORBはJava同様にインタプリタ言語であるが、実時間性を必要とするアプリケーションではインタプリタの処理速度が問題となる場合が考えられる。

2.2 プラットフォームの要求仕様

以上の観点を踏まえ本研究では、次に挙げるような機能をアプリケーションに提供できるプラットフォームを考える。

- (1) 異機種の分散ハードウェア環境にアプリケーションを構築/運用できる。
- (2) クライアント/サーバ形とピアトゥピア形のアプリ

Fig.1 Configuration of the decentralized autonomous system platform and implementation of sub-systems

ケーションを構成できる。

- (3) 同じハードウェア環境で複数のアプリケーションを独立に構築/運用/管理できる。
- (4) サブシステムの名前を指定するだけでその実装位置に関係なくメッセージを送信できる。
- (5) サブシステムの実装位置を分散ハードウェア環境の中で移動できる。
- (6) サブシステムを結合したり、結合させたサブシステムを分離できる。

プラットフォームのプロトタイプは、UNIXをOSとするEWSが提供する分散ハードウェア環境を対象に、ネットワークプロトコルにはTCP/IPを用い、アプリケーションはC言語で記述するものとして開発する。対象をC言語とするのは、汎用言語を利用するという枠の中で、上記の仕様を満足するプラットフォームの実現がどの程度可能であるか評価するためである。

3. アプリケーションプラットフォーム

3.1 プラットフォームの構成

図1に各EWS上のプラットフォームの構成とサブシステムの実装形態を示す。プラットフォームは、ブローカとインタプリタコアから構成される。これらはプロセスとしてそれぞれEWS上で動作するプログラムである。サブシステムはインタプリタコアへリンクする形で実装する。インタプリタコアへは複数のサブシステムをリンクできる。アプリケーションは、サブシステムをこのような形態で複数のEWSへ分散/実装することにより構築する。

3.2 ブローカ役割

ブローカは、ブローカデータとネーミングデータを管理することにより、アプリケーション毎に利用可能な分散ハードウェア環境と実装位置に透過なサブシステム間通信機能を提供する。ブローカデータは、同一のアプリケーションに属するサブシステムが利用可能なEWSのアドレス情報である。ここに書かれたすべてのEWSがアプリケーションの実行環境となる。自律分散システムでは、サブシステムの相互作用による動作がシステム全体の挙動を決定する。そのため、異なるアプリケーション間では、通信による情報交換などサブシステム間の干渉を避けたい場合がある。このような場合は、それぞれに対しブローカを起動しブローカデータを設定すれば、同一のハードウェア上で複数のアプリケーションを独立に管理できる。

ネーミングデータはサブシステムの実装位置と名前である。サブシステムはお互いを識別できるように以下に示す規則にし

Fig.2 Relation between sub-system naming rule and logical structure of sub-systems

たがって名前をつける。

Root/Class name/Class name/.../Sub-System name

サブシステムは、図2に示すように、“/”により区切られた文字列で表す階層関係をもつものとして管理する。“Class name”はサブシステムが属するクラス名，“Sub-System name”は実体名である。この名前と関係は、アプリケーションの構造にしたがって設計者が設定する。

ブローカはブローカデータとネーミングデータからサブシステムの実装位置を参照し指定された相手へメッセージを配信する。これにより名前を指定するだけで特定のサブシステムへメッセージを送信することが可能となり、サブシステムの移送や結合、分離などによるサブシステムの実装先や構成が変化しても、サブシステムの通信部分のプログラムを変更しないで済む。また、クラス名を指定するなどサブシステムの論理的な関係を利用することで1対多の通信が可能となる。

3.3 インタプリタコアとサブシステム

インタプリタコアはEWS上で起動できるプロセスの数の制限内であれば幾つでも起動することができる。ひとつのインタプリタコアには複数のサブシステムをスレッドとしてリンクできる。これにより、EWSで起動可能なプロセスの数の制限を受けることなく複数のサブシステムを1台のEWS上へ実装できることになる。

サブシステムは、図3に見るように、トランザクションの集まりとして記述する。インタプリタコアはサブシステムに送られてくるメッセージに対して対応するトランザクションを取り出し実行するというイベント駆動エンジンである。複数のサブシステムをリンクしているインタプリタコアは、イベントの単位で各サブシステムへの実行制御のサービスを切り替える。インタプリタコアはメッセージに対してトランザクションを取り出すために、メッセージとトランザクションの関係を記述したテーブルを参照する。このテーブルをメッセージ/トランザクション連合ルールと呼ぶ。

イベントとしてトランザクションの処理を完結させ、サブシステムの実行制御部分をインタプリタコアとして切り離し、トランザクションをテーブルを介して間接的に実行する形式をとることにより、トランザクションをインタプリタコアへリンクしテーブルを書き換えるという単純な操作でトランザクションの追加、置き換え、削除が可能になる。

3.4 アプリケーションプログラムインタフェース

メッセージ通信、サブシステムの移送、結合および分離などの機能は、関数として呼び出し実行できるようにする。これらの関数はインタプリタコアにあらかじめリンクしてある。サブシステムは自動的にコンパイルされインタプリタコアへリンクさ

Fig.3 Configuration of sub-system and control mechanism of the Interpreter Core

```

① Static JOINTFUN FuncTable[]={
    {"CallSubSystem",PreCallSubSystem,CallSubSystem,PostCallSubSystem},
    {"TimerProg",PreTimerProg,TimerProg,PostTimerProg},
    {"",NULL,NULL,NULL};
}

② AGENTLIST AgentID=
    {0,0,"C11/S1",0,"",-1,NULL,FuncTable,NULL,NULL,NULL};

//Transaction for message "CallSubSystem"
int PreCallSubSystem(int argc,char *argv)
{
    Program code of function "PreCallSubSystem"; ③
    return( TRUE );
}

int CallSubSystem(int argc, char *argv)
{
    Program code of function "CallSubSystem"; ④
}

int PostCallSubSystem(int argc,char *argv)
{
    Program code of function "PostCallSubSystem"; ⑤
}

//Transaction for message "TimerProg"
int PreTimerProg(int argc,char *argv)
{
    Program code of function "PreTimerProg";
    return(TRUE);
}
    
```

Fig.4 A coding example of sub-system

れるので、単に関数を呼び出すコードをトランザクションのプログラムの中に記述すればこれらの機能を使うことができる。また、広域の分散ハードウェア環境へのサブシステムの実装やトランザクションの追加、置き換え、削除などのメンテナンスを遠隔地からも行えるように、移送、結合、分離機能はメッセージの送信という形式でリモートに実行できるようにした。

4. アプリケーションフレームワーク

4.1 サブシステムの記述

サブシステムを構成するトランザクションは、図3に示すように、プリプロセス、プロセス、ポストプロセスと呼ぶ3つの関数から構成される。プリプロセス、プロセス、ポストプロセスはC言語でコーディングする。図4にサブシステムの記述例を示す。図中①はメッセージ/トランザクション連合ルールの記述例である。メッセージテキストの後にプリプロセス、プロセス、ポストプロセスのエントリーポイントを対応づける。図中②は、インタプリタコアがサブシステムを実行制御する際に参照する情報である。“C11/S1”はサブシステムの名前である。ここで“C11”はこのサブシステムが属するクラス、“S1”はその実体名である。“NULL”の部分には、サブシステムが起動する際インタプリタコアが自動的に必要な情報を記述する。図中③、④、⑤にはプリプロセス、プロセスおよびポストプロセスのコードを書く。トランザクションへの引数はファイルにリストして渡す。渡される引数は各関数に共通である。各関数はファイルから処理に必要な引数を読み込む。引数を記述したファイルの名

前は図に見るような形式で関数へ渡す。これは関数を記述する際の規則である。引数をファイル渡しとするのは、CPUに依存する機種特有の処理をプラットフォームにもたせないこと、大きなサイズの引数の扱いや他の言語とのインタフェースを簡略化するためである。

4.2 サブシステムの実行

トランザクションの実行では最初にプリプロセスを実行する。プリプロセスは真か偽かを返す関数としてコーディングする。プリプロセスの実行結果が偽のとき、トランザクションの実行はここで終了する。真であれば、次にプロセスを実行し、続いてポストプロセスを実行する。トランザクションの中で関数が定義されていない場合、インタプリタコアは未定義関数を飛ばして次の関数を実行する。例えば、プリプロセスが未定義の場合は、インタプリタコアはプロセスを最初に実行する。

このインタプリタコアによるトランザクションの処理シーケンスを利用することにより、さまざまな実行形態のサブシステムを構築することができる。例えば、プロセスだけを定義したトランザクションからなるサブシステムは、メッセージに対してトランザクションが反応する単純なイベント駆動システムとなる。また、プリプロセスをサブシステムの状態評価、プロセスをメッセージに対する反応、ポストプロセスをサブシステムの状態遷移関数とすることで、簡単なプロダクションシステムやオートマトンを構成することもできる。さらに、プロセスの実行結果にしたがって自分自身にメッセージを送る関数としてポストプロセスを定義することにより、トランザクションの実行を自律的に制御できるサブシステムを実現できる。

4.3 サブシステムの移送

図5にサブシステムの移送処理の流れを示す。プラットフォーム内のインストラクションをイベント駆動形式で統一するために、サブシステムはメッセージとして移送先へ送る。メッセージには、図中①に示すように、トランザクションのオブジェクトコード、メッセージ／トランザクション連合ルール、未処理のメッセージ、サブシステムの中で定義されているグローバルデータをコピーする。このメッセージは、図中②に示すように、他のメッセージ通信と同様にブローカを介して移送先のインタプリタコアへ送る。このメッセージを受け取ったインタプリタコアは、図中③に示すように、メッセージ／トランザクション連合ルールをメモリ上へ展開する。次に、オブジェクトコードをダイナミックリンク機能を使って自分へリンクする。最後に未処理のメッセージをメッセージ格納場所へコピーし、グローバルデータの内容をシンボルテーブルを参照して所定の場所へコピーする。このように移送直前のグローバルデータを復元することによりサブシステムは移送後もそれまでの処理結果を継続できる。

移送先のコンピュータの機種が移送元のものと異なる場合は、オブジェクトコードではなくソースコードを送る。ソースコードはリモートコンパイルし機種に合ったオブジェクトコードを生成した後、インタプリタコアへリンクする。移送先のインタプリタコアを特定しない場合は、ブローカが移送先のコンピュータ上に新たなインタプリタコアを起動しそこへサブシステムを移送する。

4.4 サブシステムの結合と分離

サブシステムの結合は、2つのサブシステムのトランザクションをインタプリタコアへリンクしメッセージ／トランザクション連合ルールを1つに結合することにより実現する。移送の際

Fig.5 Procedure for sub-system transportation

Fig.6 Logical structure and implementation of the example application

と同様に結合するサブシステムはメッセージとして結合先のサブシステムの実行を管理しているインタプリタコアへ送られる。このメッセージを受け取ったインタプリタコアは、メッセージの中に記述されているトランザクションのオブジェクトコードをダイナミックリンク機能を使ってインタプリタコアへリンクする。次に、移送されてきたサブシステムのメッセージ／トランザクション連合ルールを結合先のサブシステムのそれに追加する。2つのサブシステムのメッセージ／トランザクション連合ルールに重複がある場合、すなわち同一のメッセージに対して2つの異なるトランザクションが対応する場合には、後に結合したトランザクションを有効にし前のものは無効にする。サブシステムは自分自身あるいは自分の完全なコピーを他のサブシステムへ結合できる。

結合したサブシステムは、再び元の2つのサブシステムに分離できる。分離では、はじめにメッセージ／トランザクション連合ルールを分離し、次に分離したメッセージ／トランザクション連合ルールとトランザクションのオブジェクトコード、グローバルデータからなるメッセージを作成する。それをサブシステムの移送手続きにしたがって他の場所へ移す。この際、結合によって無効になったトランザクションは分離によって再び有効となる。また、グローバルデータの内容は、結合中に更新されたものを継承する。結合したサブシステムがコピーの場合、同じサブシステムが複数できることになるので、分離後のコピーは他の場所へ移送せず消去する。

5. アプリケーションの実装例

5.1 アプリケーションの構成

開発したプラットフォームを使ったアプリケーションの実装例として、ドリル加工のNCコードを生成する簡単なシステムを構築した。アプリケーションは、ドリル加工に用いる工具を表す Drill_1, Drill_2, Drill_3 と呼ぶ3つの工具サブシステムと Work_1, Work_2 と呼ぶ工作物サブシステムからなる。各サブシステムは、図6に示すように、DrillクラスとWorkクラスに属す

Fig.7 Result of the NC code generation system constructed on the developed platform

るものとして名前付けする。工作物サブシステムは、加工したい穴の位置と径のデータをもつ。工具サブシステムは、それぞれ異なる径のドリル工具に対応しており、指定された穴位置に指定された深さの穴加工を行うためのNCコードの生成と工具経路を表示するトランザクションをもつ。計算環境は、ネットワークで結合しているEWSHP9000 Series 400とHP9000 Series 700である。工作物サブシステムはHP9000 Series 400上へ、工具サブシステムはHP9000 Series 700上へ実装する。

5.2 アプリケーションの動作結果

例題アプリケーションでは、工作物サブシステムは自己のタスクである穴加工を行うために工具サブシステムのトランザクションを利用しなければならない。これは、協調問題解決の簡単な例と見ることができる。一般に協調問題解決では、情報交換を基礎とした協調の仕組みを議論の対象とするが、ここでは、この問題には立ち入らず、実装/運用環境を評価するという立場から、簡単なメッセージの交換によってサブシステムの移送、結合、分離機能を起動し、与えられたタスクを実行できるように工作物サブシステムの機能を自律的に拡張させるアプリケーションを考える。

アプリケーションの動作は、工作物サブシステムが穴情報を工具サブシステムへ送り、工具サブシステムはその穴情報に反応して自分のコピーを工作物サブシステムへ移送し結合する。工作物サブシステムは、結合した工具サブシステムのトランザクションを使ってNCコードの生成と工具経路の表示を行う。図7は各サブシステムが別のインタプリタコアへ実装されている場合の実行画面である。工作物サブシステムが移送と結合により工具サブシステムのNCコード生成機能を獲得し、それを使ってNCコードと工具経路を生成/表示したことを確認した。

例題アプリケーションで行った工作物および工具サブシステムの結合は、工作物サブシステムへの新たな機能の追加と見ることができる。同様に分離は機能の削除に、結合と分離を組み合わせれば機能の入換えになる。自律分散システムでは、変化に対する柔軟性が求められるが、移送や結合、分離機能を使えば、サブシステムの動作を停止することなく、サブシステムの機能の追加、削除、入換えができ、問題状況の変化に対してアプリケーションの構成や機能を柔軟に変えることができる。ネットワーク内を移動しながら目的の処理を実行するモバイル/ネッ

トワークエージェントの研究⁸⁾が行われているが、開発した移送機能を使えばネットワークを移動しながら目的の処理を実行するサブシステムを構成できる。さらに、結合や分離機能を使えば、自己の機能を拡張しながら自律的に動作するサブシステムの実現も期待できる。

6. おわりに

分散ハードウェア環境上にアプリケーションを実装/運用するプラットフォームの開発を行い次のような結論を得た。

- (1) EWSにより構成する分散ハードウェア環境のもとでのアプリケーションの構成方法を示し、それに基づくプラットフォームを開発した。
- (2) システムの柔軟性を向上させる機能として、サブシステムの移送と結合および分離を挙げ、オペレーティングシステムが提供するシステム関数を用い開発した。
- (3) 開発したプラットフォーム上に簡単なアプリケーションを構築しプラットフォームの有効性および効果を確認し、プラットフォームがシステムのスケラビリティの向上にどのように寄与し得るかの一例を示した。

参考文献

- 1) R.G.Smith: The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver, IEEE Trans. Computers, c-29, 12 (1980) 1104.
- 2) 森 欣司, 宮本捷二, 井原廣一: 自律分散概念の提案, 電気学会論文誌, 104, 12 (1984) 303.
- 3) 田村信介, 岡宅泰邦, 関 俊文: 知的分散システムのアーキテクチャ, 電気学会論文誌C, 108, 6 (1988) 393.
- 4) 塚本享治: オブジェクト指向分散環境OZ++の基本設計, システムソフトウェアとオペレーティングシステム, 61, 3 (1993) 17.
- 5) B.C.Johnson: Distributed Computing Environment Framework: An OSF Perspective, <http://www.osf.org>, (1991).
- 6) R.Otte, P.Patrick, and M.Roy: UNDERSTANDING CORBA, Prentice Hall, (1996), (邦訳「分散オブジェクト指向CORBA」, プレンティスホール, (1996)).
- 7) S.Hirano: HORB, <http://ring.etl.go.jp/openlab/horb/>, (1996).
- 8) 永井保夫, 田原康之, 入江 豊, 大須賀昭彦, 本位田真一: Plangent I: インテリジェント・ネットワークエージェント, <http://www2.toshiba.co.jp/plangent/indexj.html>, (1996).