Vol.2009-ARC-183 No.29
Vol.2009-OS-111 No.29
2009/4/24

抽象ネットワークによるシステム記述

中 村 和 敬　　日 比 野　靖[†1]

現在のコンピュータシステムではネットワーク環境が一般化している。このような環境ではシステムを外部ネットワークや IPC 、内部バス等から成るネットワークとして記述し、機能を拡張したスイッチとして扱うことによって、分散形態の自由度向上、並列制御フローなど、ネットワーク分散システムに有益な様々な利点を得ることが出来る。本論分ではそのような方針に基づき決定されたシステム記述言語仕様について述べ、その記述能力を検討する。

A System description language by Abstract Network

Nakamura Kazutaka and Hibino Yasushi[†1]

Nowadays, network computing environment have became popular. In such environment, a language describes system as network consisting of communication channels, those are outer network, IPC and inner bus, and function as enhanced switch, and provides various advantage on distributed network system, those are flexibility of distributed style, parallel control flow and etc. In this article, It is described and evaluated that system description language is developed from such view.

1. はじめに

1.1 Element Computing

現在、どこでもコンピュータ、どこでもネットワークというユビキタス社会が実現されつつある。今、個々のコンピュータを連携して統合運用するネットワーク分散システムが必要である。そしてそれは最低限の知識さえあればどんな利用者で扱えるものである事が望ましい。すなわち “誰でも分散システム” が必要である。

現在までにいくつかのネットワーク分散システムが開発されている。しかしながらそれらの多くは各々のアプリケーション毎に開発されている為バラバラのインターフェースを持っておりユーザが導入する際の負担となっている。(例: IRC: チャットアプリケーション,NFS: ファイルシステムアプリケーション) また同時に、同じ用途のアプリケーションであっても、個々のアプリケーション毎に固有の分散形態をもっており、柔軟性に欠ける。(例: Samba: ファイルシステムレベルでの分散, NFS: デバイスドライバレベルでの分散) この様にアプリケーション毎の分散システムは、必要に応じて自在に分散システムを利用する為の障害となっている。汎用的で、どのような分散形態でも実現できる分散システムが必要である。

少数ながらそのような分散システムも開発されている。CSP(Communicating Sequential Processes[1]) モデルやメッセージパッシングモデルに基づく分散 OS、RPC(Remote Procedure Call) 機構に基づく分散処理機構がそうである[4)–6)]。しかしながらそれらは P2P ネットワーク環境でない、従来のマスタースレーブ型の通信機構に基づいた構造になっている。現在の分散環境で一般的に用いられるクライアントサーバモデルはマスタースレーブ、コールリターンをそのままネットワークワイドに拡張したものである。この時通信の流れは常に呼び出し元に戻るので、ハブを中心としたスター型となる。これは分散形態を拘束し、並列計算の妨げとなるばかりでなく、大きな通信オーバーヘッドを生む元凶となる。新しい分散システムは、入力元と出力先が固定されず、並列計算モデルである、データフロー方式[2),7)] であることが望ましい。

以上を踏まえて筆者らが提案する Element Computing[9] は、ネットワークに分散して存在する機能を Element と呼び、それらを結合して新しい機能を実現する為の計算環境を提供するフレームワークである。この結合はデータフロー方式によって行われる。であるので、Element Computing はデータフロー指向ネットワーク分散システムの一種である。これは情報技術の知識のない専門外の人々にとっても理解しやすく使いやすいものになり、また専門家にも分散方式の柔軟性を性能を殆ど落とすことなく提供できるだろう。同時にこのネットワーク分散システムは単なるネットワーク分散システムではなく、スタンドアロンのシステムをもより柔軟に取り扱う事ができるので、実装の枠を越えたシステムとなるだろう。

1.2 Abstract Network

Element Computing の観点では、コンピュータシステムは様々なネットワークによって結合されたシステムコンポーネントの集合と考える。ここでの “様々なネットワーク” とは、システムコンポーネントが通信に用いるもの全てであり、PCIbus、EtherNet、USB のよ

†1 北陸先端科学技術大学院大学
Japan Advanced Institute of Science and Technology

Vol.2009-ARC-183 No.29
Vol.2009-OS-111 No.29
2009/4/24

うな物理的なものから、関数呼び出し、システムコール、IPC、インターネットのような論理的なものまでを含む。“システムコンポーネント” はコンピュータシステム上で通信以外の機能を提供し、これが “様々なネットワーク” を通じて通信し合い、コンピュータシステム全体としての機能を実現している。

Abstract Network(AbNet) は Element Computing の為のバーチャルネットワークであり、Element を利用するための機能を提供する。AbNet はこの “様々なネットワーク” をデータリンク層、すなわち Abstract Datalink(AbLink) として用いて構成される。元のネットワークとデータリンクについては以下 Base Network(BaseNet)、Base Datalink(BaseLink) と呼ぶ。同時に “システムコンポーネント” とはつまり Element である。Element は AbNet の Node なので、以下の議論では Element を Node と呼んでいる。

この様にシステムを捉えているので、目的のシステムコンポーネントの結合を、ネットワーク分散環境であってもスタンドアロン環境であっても同じ方法で取り扱う事が出来る。

1.2.1 コネクション指向

AbNet はコネクション指向ネットワークであり、個々の AbNet 上のノードは AbNet 上のスイッチである。IP ネットワークはパケットフォワーディングネットワークという違いはあるものの、やはり個々のノードがスイッチである。IP ネットワークではスイッチはルータと呼ばれている。ネットワークは通常 1 入力 1 出力のデータ転送機能を提供する。AbNet では多入力多出力の計算機能を提供する。転送機能は入力をそのまま出力する計算と考える。これにより AbNet の経路がそれ自体がデータフローグラフとなり、新たに構成した機能となる。AbNet はコネクションセットアップ時に全てのアドレスを BaseNet のものに変換する。従来のミドルウェアなどによる分散システムでは、常にそのオーバーヘッドが発生していたが、AbNet ではこの方式によりデータ転送時のオーバーヘッドは存在しなくなる。ミドルウェアでも、対症療法的な内部処理のショートカットなどで最適化を図っていたが、AbNet ではこれは経路探索という概念で一般化して取り扱われ、自動的に解決できる可能性を開く。これにより新しい通信方式などに即座に対応する事が可能となる。

本文ではこの AbstractNetwork つまり抽象ネットワークの仕様、及び記述方法を解説し、その記述能力を検討する。

2. Abstract Network の構成要素

以下では Abstract Network の構成要素を解説する。

AbNet の要素は Node と Link に分けられる。AbNet A は Node の集合 N と Link の集合 L の組として書き表される。

$$A = (N, L)$$

2.1 Link

Link は単方向データ転送を行い、Node 間を結ぶ。同じ Node の組の間でも方向が異なれば別の Link として扱う。一つの Link の二つの端点が同じ Node であってもよい。また同じ Node の組の間に複数の Link が存在してもよい。

以上から分かるように、AbNet は有向多重グラフによって書き表される。であるので、各 Link l に始点 $init(l)$ と終点 $ter(l)$ を割り当てる二つの写像

$$init : L \to N, ter : L \to N$$

が定められている。$init(l) = n$ かつ $ter(l) = m$ なる Link l は、n から m への単方向データ転送を行う事ができる。

ここで、ある Node に対して入力方向にある Link を InputLink(以下 ILink)、出力方向にある Link を Output Link(以下 OLink)、二つの端点が同一の Node である Link を Local Link(以下 LLink) と呼ぶ。これは Node n を端点とする Link l が存在するとき、$ter(l) = n$ ならば l は n の ILink、$init(l) = n$ ならば l は n の OLink、$ter(l) = init(l) = n$ ならば l は n の LLink であると言うことである。

2.2 Node

全ての Node は AbNet のアドレスを持っている。Node は幾つかの ILink から入力されたデータを受け取る。また、内部の関数に従っていくつかの OLink に出力する。

2.2.1 Slot

各々の Node は、どの ILink/OLink を入力/出力として扱うのかを記憶しており、これを Slot と呼ぶ。Input Slot と OutputSlot(以下 ISlot/OSlot) の区別があり、各々の Slot は記憶できる Link に制限を設ける事が出来る。ISlot には ILink のみが記憶され、OSlot には OLink のみが記憶される。通常 1 つの Slot あたり 1 本の Link を記憶する事が出来る。

Slot に Link を記憶させる操作を Link 割り当てといい、後述する Router が行う。全ての Node は自身に対して Link 割り当て操作を行う Router を持っている。ただし実際に割り当てが出来るかどうかは Node によって不定である。

各々の Slot には Node 毎に独立したアドレス空間での Slot Address 割り当てられる。Slot は ISlot、OSlot 毎に順序があり、列挙することができる。

以上から Node n は関数 f と ISlot の集合 $\langle ISlot \rangle$ と OSlot の集合 $\langle OSlot \rangle$、及び各種操

Vol.2009-ARC-183 No.29
Vol.2009-OS-111 No.29
2009/4/24

作を行う Router r_o の組として書き表される。

$$n = (f, r_o, \langle ISlot \rangle, \langle OSlot \rangle)$$

Slot s は割り当て可能な link の制約 r_s と割り当てられている link の集合の組として書き表される。

$$s = (r_s, \langle AssignedLink \rangle)$$

Node n の Slot s を $[n, s]$ と記述する。

3. Abstract Network の振る舞い

以下では Abstract Network の個々の構成要素がどのように振る舞い、またそれらが強調することによってどのような機能を如何にして提供するのか説明する。

AbNet は一組の OSlot と ISlot の組の間を結ぶ適切な経路を発見し、経路上の Node が適切な入出力を行うことにより、任意の Node 間の転送/計算機能を提供する事ができる。ただし、経路を発見できない場合や Node が適切な入出力を行えない場合はこの限りではない。これはつまり Node 間でのデータ転送中に所望の入出力対応関数を通る事により、任意の Node 間での計算が行われるという事である。通常の転送機能はデータの改変なく入力をそのまま出力する場合として捉えられる。

要求される経路 p は経路に要求される処理関数 f と入り口である OSlot 集合 $\langle OSlot \rangle$ と出口である ISlot 集合 $\langle ISlot \rangle$ の組で表される。

$$p = (f, \langle OSlot \rangle, \langle ISlot \rangle)$$

発見され、Link 割り当てが行われた接続 c は経路上の Node 集合 $\langle Node \rangle$ と Link 集合 $\langle Link \rangle$ と割り当て $\langle Assignment \rangle$ の組で記述される。

$$c = (\langle Node \rangle, \langle Link \rangle, \langle Assignment \rangle)$$

3.1 Link の振る舞い

Node は Link をバイトストリームとして取り扱う事が出来る。

3.2 Node の振る舞い

Node は幾つかの ILink から入力を受け取り、内部の関数に従っていくつかの OLink に出力する。Node は通常 ISlot に割り当てられている ILink からの入力を受け取り、ISlot に割り当てられていない ILink からの入力を破棄する。同時に OSlot に割り当てられている OLink へ出力し、OSlot に OLink が割り当てられていなければその出力を破棄する。大抵の Node は自身の任意の OSlot からの出力を任意の ISlot に入力する LLink を持っている。

Node の振る舞いを制御する為に、いくつかの操作がある。

図 1 要求された接続

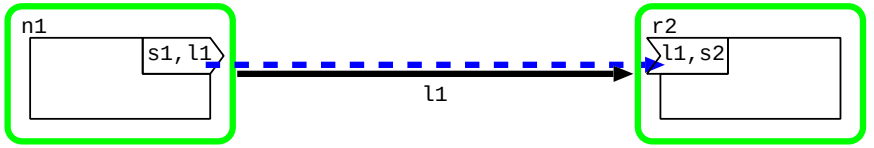

図 2 直接接続

初 期 化

Node を初期化する。Node が状態を保持するような関数に因って動作している場合は初期状態に戻る。これは後述する Router によって行われる。

Link 割り当て

Slot に Link を割り当てる。割り当ては Slot 毎に行われる。全ての Slot に Link が割り当てられておらずともよい。

$[n, s]$ に Link l を割り当てる操作を、$[n, s] \leftarrow l$ と記述する。

3.3 接 続 例

以下ではどのように接続を確立するか、例をあげて説明する。

例えば図 1 の様な OSlot $[n_1, s_1]$ と ISlot $[n_2, s_2]$ の間の転送機能 $Through$ を提供できる経路を探索しているとすると、この要求経路 p は、

$$p = (Through, \{[n_1, s_1]\}, \{[n_2, s_2]\})$$

となる。ここで $init(l_1) = n_1$ かつ $ter(l_1) = n_2$ なる Link l_1 が存在し、l_1 が他の Slot に割り当てられていなければ、

$$c = (\emptyset, \{l_1\}, \{[n_1, s_1] \leftarrow l_1, [n_2, s_2] \leftarrow l_1\})$$

の様な接続 c を確立する事が出来る。これは図 2 の様な形となる。これにより AbNet は $[n_1, s_1]$ から $[n_2, s_2]$ への転送機能を提供できる。

また、その様な l_1 が存在しない場合でも、ISlot i_1 からの入力を OSlot o_1 にそのまま出

Vol.2009-ARC-183 No.29
Vol.2009-OS-111 No.29
2009/4/24

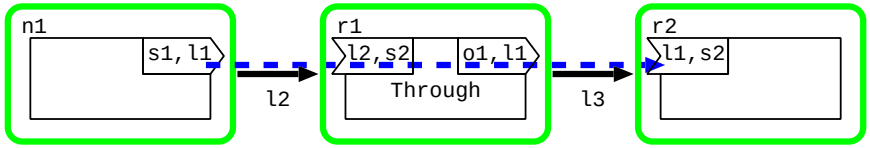

図 3 中継接続

力する *Through* 関数によって制御される Node

$$r_1 = (Through, Router, \{i_1\}, \{o_1\})$$

が存在し、$init(l_2) = n_1, ter(l_2) = r_1$、$init(l_3) = r_1, ter(l_3) = n_2$ なる割り当てられていない Link l_2、l_3 が存在するならば、

$$c = (\{r_1\}, \{l_2, l_3\}, \{[n_1, s_1] \leftarrow l_2, [r_1, i_1] \leftarrow l_2, [r_1, o_1] \leftarrow l_3, [n_2, s_2] \leftarrow l_3\})$$

という割り当てを行うことによって、図 3 に示されるような Slot 間の転送機能を提供できる。適切な中継 Node が存在するならば、この様な中継は何段でも可能である。

また $[n_1, s_1]$ と $[n_2, s_2]$ の間で何らかの演算が行われる必要があるならば、適切な処理を行う関数を持っている Node を r_1 の位置に割り当てることによって、接続は計算機能を提供する事が出来る。

また、N 個の OSlot から M 個の ISlot の間で何らかの演算機能が必要な場合も、しかるべき関数を持った多入力多出力の Node を r_1 の位置に割り当てることによってやはり計算機能を提供できる。

3.4 システム Node

Node には以下に示すように AbNet それ自体の機能を提供する物が存在する。これら AbNet の機能を利用して個々の Node の機能の間に経路を設定し、それにより新しい機能を構成する。

3.4.1 Router Node(Router)

初　期　化

Router は指定された Node の初期化を行う。

コネクションセットアップ (経路探索/Link 割り当て)

Router は Node 間の経路の発見、および経路上の Node の Slot への Link 割り当てを行う。経路が存在しなかったり、経路上に適切な関数を保有する Node が存在しない場合には経路探索は失敗する。

Router は全ての Node と Link の関係の他、Node 毎の AbNet Address、Slot、対応する router などの情報を保持している。AbNet Address、Slot Address は Link 割り当て時にのみ用いられ、データ転送時には用いられない。

4. Abstract Network 記述言語

以下では Abstract Network をどのように記述するか解説する。

シンタックスには S 式の形式を用いる。AbLink の名前空間と Node の名前空間がある。これらの記述は Router が保持する。

4.1 Link: BaseNet

Link を実際に提供するのは BaseNet である。リンクの記述は BaseNet 単位で行う。

AbNet の記述では Link を一つ一つ列挙するのではなく、BaseNet 単位で記述し、その BaseNet に参加する Node に Link が提供されていると仮定する。

```
<ablinks> :=
(basenet <name> ; <basenet-name>
  (protocol <name>) [(tranceport <name>)]
  [(links <link> ...)] [(nonlinks <link> ...)] )
```

<basenet-name>は BaseNet の名前である。protocol 節はその BaseNet のレイヤのプロトコルである。transport 節はその BaseNet でトランスポート層の機能が利用可能で有れば記述され、そのプロトコルが記述される。links 節では存在する link を記述する。links 節がない場合全ての Node の間に Link が存在すると仮定する。nonlinks 節の記述では links 節で記述された link の内存在しない link を記述する。nonlinks 節がない場合 links 節で記述された全ての link が存在すると仮定する。また以上から分かるように links 節の記述と nonlinks 節の記述が矛盾する場合 nonlinks が優先される。

```
<link> := (<addr> <addr>) ; (<srouce address> <destination address>)

<addr> := <network-addr> | (<network-addr> <transport-addr>)
```

Link の記述は BaseNet アドレスの組によって記述される。

4.2 Node

Node の記述は以下の様に行う。

```
<node> :=
(node <name> ; <node-name>
  (basenet <address> ...) ; <basenet-address>
```

Vol.2009-ARC-183 No.29
Vol.2009-OS-111 No.29
2009/4/24

```
  (router <name> ...) ; <router-name>

  [(slot <direction> <name> [<restriction>...]) ...]

  (function <function>) )

<address> := (<name> <addr>) ; (<basenet-name> <basenet-node-address>)
```

<node-name>は node の名前である。basenet 節にはその Node がどの BaseNet のどのアドレスに位置しているかを記述する。<basenet-name>に local が現れる場合、その Node は自身の全ての任意の OSlot と全ての任意の ISlot を結合する事が可能な Link を持っている事を意味する。router 節にはその Node のコネクションのセットアップを行う router を記述する。slot 節には存在する Slot について記述する。function 節にはその Node の入出力を制御する関数を記述する。

```
<direction> := in | out

<restriction> := local | (basenet <name> ...) | (links <link> ...)
```

その Slot に割当て可能な Link を指定することができる。記述がない場合は全ての Link が割当て可能であるとされる。basenet 節は BaseNet 単位で制限を設けるために利用する。links 節の記述では個別の Link 単位で制限を設けることが出来る。

```
<link> := (<name> <addr> <addr>)
; (<basenet-name> <source-address> <destination-address>)
```

4.3 操作記述

AbNet に対する操作は以下のように記述される。

```
<initialize> := (init <name>) ; <node-name>
<connect> := (connect <function> (<slot-address> ...) (<slot-address> ...))
; (connect <function> (<source-slot> ...)(<destination-slot> ...)
<assignment> := (assign <slot-address> <tlink>)

<slot-address> := (<name> <name>) ; (<node-name> <slot-name>)
<tlink> := (<name> <link>) ; (<basenet-name> <link>)
```

<initialize>は<node-name>で表される Node に対する初期化操作である。<connect>は (<function>,<source-slot>...,<destination-slot>...) なる経路による接続を試行する操作である。これらは最終的には次に述べる<assignment>に展開されなければならない。<assignment>は<slot-address>に対して<link> を割り当てる操作である。

5. 記述能力の検討

以下では
UNIX の ed(1) の様な簡単なエディタシステムを例に取り、記述能力を検討する。

5.1 Abstract Network の記述

以下のような AbNet を考える。

```
(basenet inet (protocol IP)(transport TCP))
(basenet machine1-IPC (protocol IPC)(transprot IPC-Port))
(basenet machine1-IPC (protocol IPC)(transprot IPC-Port))

(node file1  (network (machine1-IPC file-procid))
             (router TheRouter) (function FILE)
  (slot in  data-in) (slot out data-out) )

(node editor1 (basenet (machine1-IPC editor1-procid))
              (router TheRouter) (function EDITOR)
  (slot in  data-in)   (slot out data-out)
  (slot in  command-in)(slot out command-out) )

(node editor2 (basenet (machine2-IPC editor1-procid))
              (router TheRouter) (function EDITOR)
  (slot in  data-in)   (slot out data-out)
  (slot in  command-in)(slot out command-out) )

(node inet-stack1
  (network (machine1-IPC inet-stack1-procid) (inet inet-addr1))
  (router TheRouter) (function SWITCH)
  (slot in  data-in1) (slot in  data-in2) (slot in  data-in3)
```

Vol.2009-ARC-183 No.29
Vol.2009-OS-111 No.29
2009/4/24

```
  (slot out data-out1)(slot out data-out2)(slot out data-out3) )

(node inet-stack2
  (network (machine2-IPC inet-stack2-procid) (inet inet-addr2))
  (router TheRouter) (function SWITCH)
  (slot in  data-in1) (slot in  data-in2) (slot in  data-in3)
  (slot out data-out1)(slot out data-out2)(slot out data-out3) )

(node keyboard1 (network (machine1-IPC kbd1-procid))
  (router TheRouter) (function KEYBOARD) (slot out output) )

(node keyboard2 (network (machine2-IPC kbd2-procid) )
  (router TheRouter) (function KEYBOARD) (slot out output) )

(node display1 (network (machine1-IPC disp1-procid) )
  (router TheRouter) (function DISPLAY) (slot in  input) )

(node display2 (network (machine2-IPC disp2-procid))
  (router TheRouter) (function DISPLAY) (slot in  input) )
```

これを図示すると図 4 の様になる。

5.2 要求アプリケーション

この時以下のようなアプリケーションシステムを必要としているとする。

```
(node file (function FILE) (slot in  data-in) (slot out data-out) )
(node editor (function EDITOR)
  (slot in  data-in)   (slot out data-out)
  (slot in  command-in)(slot out command-out) )
(node keyboard (function KEYBOARD) (slot out output) )
(node display (function DISPLAY)   (slot in  input) )

(connect (file data-out) (editor data-in))
(connect (editor data-out) (file data-in))
```

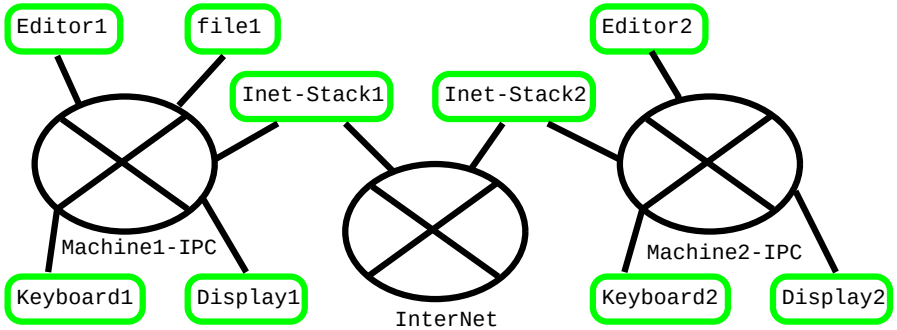

図 4 AbNet の例

```
(connect (keyboard output) (editor command-in))
(connect (editor command-out) (display input))
```

ここでの各々の Node 名は仮のものである。いま file は先ほどの記述の file1 に対応しているとする。ここで editor、keyboard、display のいくつかの対応付けを行い、正しく機能を実現出来るか、つまり結合できるかを見る。

5.3 スタンドアロンシステム

Stand Alone タイプのシステムを実現する場合を考えてみる。file は AbNet 記述から machine1-IPC という BaseNet に所属していると分かる。そこで他の Node もこの BaseNet に所属しているものを選ぶ。すると、editor→editor1、keyboard→keyboard1、display→display1 となり、接続要求は Link をそのまま割り当てる事が出来るので、

```
(assign (file data-out) (machine1-IPC ((file-procid 1) (editor1-procid 0))))
(assign (editor1 data-in)(machine1-IPC ((file-procid 1) (editor1-procid 0))))
(assign (editor1 data-out)(machine1-IPC ((editor1-procid 1) (file-procid 0))))
(assign (file data-in)(machine1-IPC ((editor1-procid 1) (file-procid 0))))
(assign (keyboard1 output)(machine1-IPC ((kbd1-procid 0) (editor1-procid 2))))
(assign (editor1 command-input)(machine1-IPC ((kbd1-procid 0) (editor1-procid 2))))
(assign (editor1 command-output)(machine1-IPC ((editor1-procid 3) (disp1-procid 0))))
(assign (display1 input)(machine1-IPC ((editor1-procid 3) (disp1-procid 0))))
```

と言う形に展開され、正しく結合することが出来た。これを図示すると図 5 の様になる。

Vol.2009-ARC-183 No.29
Vol.2009-OS-111 No.29
2009/4/24

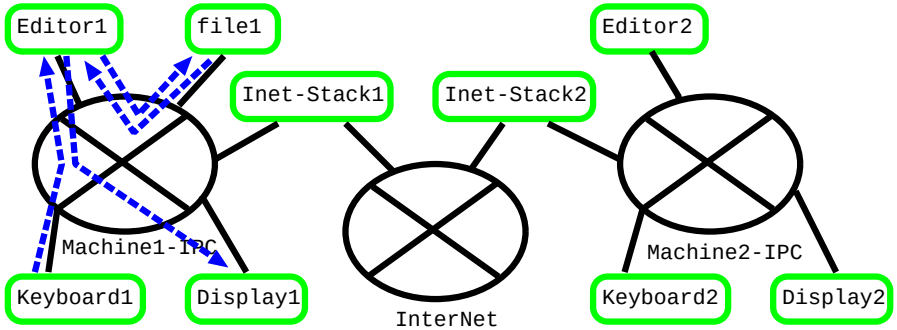

図 5 スタンドアロンシステムの例

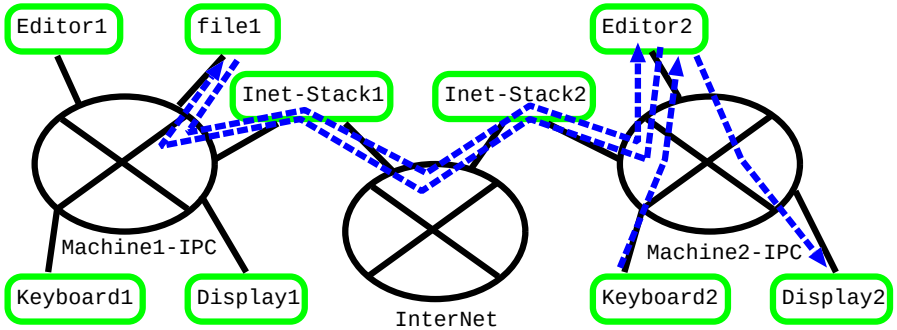

図 6 ファイルシステムレベル分散の例

5.4 ファイルシステムレベル分散

次にファイルシステムレベルで分散処理する場合を考えてみる。editor→editor2、keyboard→keyboard2、display→display2 と対応付けると、接続要求は

```
(connect (file data-out) (editor2 data-in))
(connect (editor2 data-out) (file data-in))
(connect (keyboard2 output) (editor2 command-in))
(connect (editor2 command-out) (display2 input))
```

となり、最初の 2 行の接続については中継 Node が必要となる。ここで inet-stack1 と inet-stack2 を中継 Node として用いることが出来るので、

```
(connect (file data-out) (inet-stack1 data-in1))
(connect (inet-stack1 data-out1)(inet-stack2 data-in1))
(connect (inet-stack2 data-out1)(editor2 data-in))

(connect (editor2 data-out) (inet-stack2 data-in2))
(connect (inet-stack2 data-out2)(inet-stack1 data-in2))
(connect (inet-stack1 data-out2)(file data-in))
```

と言う形に展開される。ここから先は先程と同様にそのまま Link 割り当てを行う事ができので、正しく結合することが出来る。これを図示すると図 6 の様になる。

他にも様々な分散形態が考えられるが、同様に処理することによって、様々な分散形態を実現できるとわかる。

5.5 ま と め

上では簡単なエディタシステムを例に取り、スタンドアロン環境と分散環境下での記述を行った。ここから同じ要求記述から様々な分散形態が実現可能であると分かった。これにより分散形態に囚われずアプリケーションを記述する事が可能になるだろう。

6. 今後の課題

AbNet には解決すべき課題や取り入れるべき機能がまだまだ多い。以下ではそういった課題や機能について述べる。

初期通信方法

現在 Router には適切な Link が割り当てられていると仮定している。しかしながら実用を考えるならば、Link が割り当てられていない Router と通信する為の方法を決定しておく必要がある。現在いくつかのアイディアを検討中である。

分散 Router

現在 Router は一つであり集中処理を行うと仮定している。しかしながら性能面、保安面から考えて、Router 機能を分散システムとして実現する事が必要である。

Node/Slot 生成/破棄

Node や Slot は静的に数が決まったものではなく、必要に応じて生成されたり破棄されたりするものである。これらをサポートする機能、記述が必要である。

仮 想 化

ネットワーク上の機能を上手く AbNet に組み込み、また AbNet から公開するための仮

Vol.2009-ARC-183 No.29
Vol.2009-OS-111 No.29
2009/4/24

想化機能が必要である。現在合成 Node を一つの Node として扱う Virtual Node 機能と、Node の集合から新しい Network を作り他のユーザに公開するための Virutal Network 機能を考えている。これが必要なのは、AbNet が個々のユーザのプライベートなものだからである。また、AbNet 上で組み上げたネットワークシステムを BaseNet として組み込む機能もあった方が良いかもしれない。

アプリケーション記述方法

AbNet 上のアプリケーションを記述する方法が必要である。現在 AbNet 記述の書き換えをベースのものを考えている。これは曖昧な記述を徐々に具体化していき、最終的に完全な記述となった時点でシステムとして具体化するようなものである。

性 能 記 述

自動割り当ての際には、性能が一つの尺度となる。そういった性能に関する数値を記述出来る必要がある。

様々な Element/BaseNet の為の記述方式の提供

様々な Element/BaseNet を容易に取り扱える必要がある。例えば現在 BaseNet は P2P の完全なネットワークを想定しているが、Master Slave 型のバスも取り扱える必要がある。また、そういったパターンを再利用可能な様にして、記述量を減らす仕組みも必要だろう。

物理的環境との統合

AbNet はユビキタス環境に適用するのが一般的な用途になると思われる。その為にはマンマシンインターフェースのような多様な物理的関係も BaseNet の Link として記述できないかと考えている。

7. おわりに

本文では Abstract Network の仕様について解説し、次にその記述言語を定義した。さらにその記述言語を用いて簡単なシステムを記述しその上でのアプリケーションが分散形態の違いに左右されずに実現される可能性を示した。続いて今後取り組むべき課題を述べた。

今後は課題を一つ一つ解決していく事によって、新しいネットワーク分散システムを構築していきたい。

参 考 文 献

1) C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, Vol. 21, No.8, pp. 666–677, 1978.
2) WesleyM. Johnston, J.R.Paul Hanna, and RichardJ. Millar. Advances in dataflow programming languages. *ACM Comput. Surv.*, Vol.36, No.1, pp. 1–34, 2004.
3) J.Liedtke. On micro-kernel construction. In *SOSP '95: Proceedings of the fifteenth ACM symposium on Operating systems principles*, pp. 237–250, New York, NY, USA, 1995. ACM.
4) AndrewS. Tanenbaum. 分散オペレーティングシステム. プレンティスホール出版, 1996.
5) AndrewS. Tanenbaum and RobbertVan Renesse. Distributed operating systems. *ACM Comput. Surv.*, Vol.17, No.4, pp. 419–470, 1985.
6) AndrewS. Tanenbaum and Maarten van Steen. *Distributed Systems: Principles and Paradigms (2nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.
7) PaulG. Whiting and Robert S.V. Pascoe. A history of data-flow languages. *IEEE Ann. Hist. Comput.*, Vol.16, No.4, pp. 38–59, 1994.
8) Andrew S. Tanenbaum AlbertS. Woodhull. *The MINIX book Operating Systems Design and Implementation*. Pearson Education, Inc, third edition edition, 2006.
9) 中村和敬, 日比野靖. データフロー指向ネットワーク分散システムの提案. 第 2008-OS-107 巻. 日本情報処理学会, 1 2008.