ソフトウェア工学 68－8
（１９８９．９．２６）

ソフトウェア柔構成方式としてのソフトン

岸本 芳典　　山野 紘一
㈱日立製作所 システム開発研究所

多様・大量なソフトウェア需要と分散環境上での協調実行実現に対応するために、ソフトウェアを機能要素部品ソフトン(SOFTON：SOFTware＋ON)から構成する柔構成方式を提案した。ソフトンとは、機能要素の本来の処理部である本体部と、本体部間の結合を行うコネクタ部からなるソフトウェア構成部品である。本方式では、本体部が具備するインタフェースの仕様からコネクタ部を生成し、ソフトンの組合せを記述したコネクション仕様に基づき適切なコネクタ部を用いてソフトンを結合する。これにより、結合相手に依存しない機能部品の作成と、稼働環境に適合したソフトウェア柔構成の実現が容易になる。

FLEXIBLE SOFTWARE CONSTRUCTION USING SOFTON

Yoshinori Kishimoto　Koichi Yamano
Systems Development Laboratory, Hitachi, Ltd.
1099, Ohzenji, Asao-Ku, Kawasaki, 215 Japan

We propose flexible software construction method using SOFTON to reconstruct software components and to implement distributed cooperative execution. SOFTON is a functional component, composed of a body and a connector. The body is a functional processing part, while the connector is an interface part which performs interactions between bodies. The connector is generated from interface specification of the body. SOFTONs are available to construct software system using appropriate connectors based on the specification which describes connection of SOFTONs and their environment. It is easy to make functional components which do not depend on their connected components and to reconstruct software to adapt its environment.

1．はじめに

ワークステーション、ミニコン、オフィスプロセッサ等の中小規模計算機の分野では、その適用分野、適用業務が急速に拡大し、要求されるソフトウェアは量的に増加すると共に質的に高度化、多様化している。このため、既存のソフトウェア部品を活用するソフトウェア・アーキテクチャの確立が求められている。また適用拡大にともない計算機システムの処理能力の向上も求められており、各計算機単体での能力向上だけでなく、複数の計算機をＬＡＮ接続した水平分散システム上での多様性、能力向上が求められている。

これらの要望に応えるためには、

a）再利用可能なプログラムの構成要素(部品)を使用する構成方式と部品の提供

b）分散環境上でソフトウェアを稼働させるための基盤機構の提供

が必要である。本稿では、この要望を実現するソフトウェアの構成モデルを提案する。

ソフトウェアの再利用に関する研究は従来から種々試みられているが、近年はオブジェクト指向に基づく部品化の研究が盛んである。StepStoneのSoftware-IC[1]は、Objective-Cによるクラス・ライブラリを提供する。またワシントン大のJade[2]は、Emerald言語によるプログラム部品の再利用環境である。いずれもｂ）に関しては、今後の研究課題となっている。

一方、分散環境上でのソフトウェア開発システムとしては、Xerox CedarのLupine[3]、CMUのMatchmaker[4,5]等がある。Lupineは、Mesaのインタフェース・モジュールとして記述された遠隔手続き呼出し(RPC：Remote Procedure Call)のインタフェース仕様からクライアント/サーバ用スタブを生成するシステムである。Matchmakerは、プロセス間のオブジェクト指向なRPCの仕様記述からC、Pascal、Common LISP、Ada用のRPCスタブを生成するシステムで分散プログラムの実現容易化と信頼性向上、他言語プログラムとの協調実現容易化等を目的としている。これらのシステムはいずれも上記ｂ）に対応するものではあるが、ａ）に関しては考慮されていない。

2．課　題

分散環境を考慮したソフトウェア環境では、計算機の利用者の立場からは以下のことが要望される。

（1）新たに計算機を追加導入しても、既存の計算機上で稼働する既存ソフトウェア資産を有効活用できること

（2）複数の計算機による協同処理による負荷分散・機能拡張が容易に実現できること

一方、アプリケーション開発者の立場からは以下のことが要望される。

（3）新しいアプリケーションの開発が短期で行えること

（4）アプリケーションの開発時は開発環境と実行環境とのシステム構成の差異を意識しなくてもよいこと

このような要望に対して、既存のアプリケーション・プログラムの機能要素を別のアプリケーションの機能部品として容易に利用可能にするアプリケーション・ソフトウェア・アーキテクチャの確立が重要であると考えた。そしてこのアーキテクチャに基づく豊富なソフトウェア基盤部品群を整備し、複数の機能部品を新アプリケーションの構成要素として活用できるようにする。

3．ソフトン

ソフトウェア・アーキテクチャの設計方針として以下を設定した。

（1）機能部品間のインタフェースを標準化する

（2）機能部品が複数の計算機上に分散する場合でも同一の結合方式とする

（3）具体的な結合相手や稼働計算機は、機能部品本体の外部から指定する

この方針に基づき、ソフトウェアの単位部品は、処理を行う本体部と他部品との結合を行うコネクタ部に分けることが重要であると考えた。そしてソフトウェアは、この単位部品の組合せにより構成する。この単位部品は、ソフトウェアを構成する素粒子という意味からソフトン（SOFTON：SOFTware ＋ ON；-onは構成子、素粒子という意味の接尾語）と名付けた。ソフトンの概念を図１に示す。

従来のプログラムは、図２に示すように、実行単

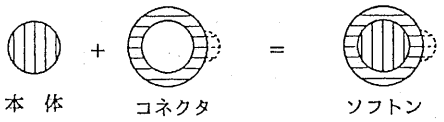


図1　ソフトンの概念

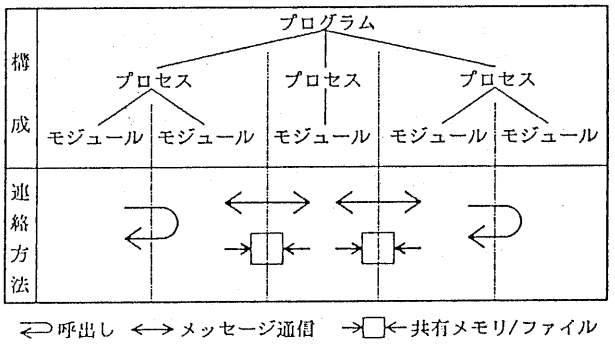


図2　従来のプログラム構成

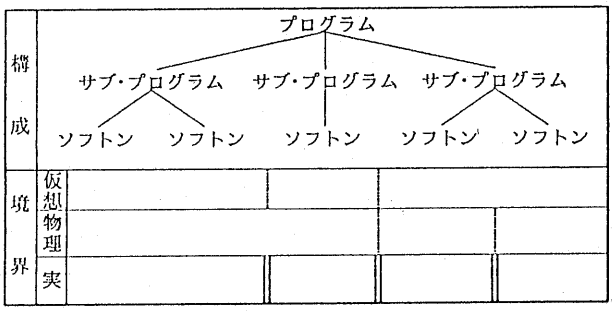


図3　ソフトンによるプログラム構成

位(OSの管理単位)であるプロセスの集合であり、各プロセスは機能分割されたモジュール(機能部品)が結合したものとみなすことができる。プロセス内でのモジュール間の連絡には関数/手続き呼出しが使用される。一方プロセス間での連絡には広義の通信が用いられ、具体的にはメッセージ通信や共有メモリ/ファイル等により情報を交換している。このようなプログラム構成方式の問題点は次の3点である。

（1）プロセス間の通信では、特定の相手と特定の通信手段で通信するためのコードがモジュール部に作り込まれている。従って他の相手と通信させたり、相手プロセスを他のマシンに分散させるためには、両者のモジュール部自体の変更が必要になる。

（2）モジュール部はモジュール間の連絡部分を含んでおり、そのためインタフェースは固定的になっている。従って他のモジュールと置き換える場合には、インタフェースの整合を取るための修正が必要となることがある。

（3）プログラムはプロセス単位にのみ分散可能である。従ってプロセス内のあるモジュールを単独で分散させるためには、プログラムの構造を変更して1つのプロセスとし、またモジュール間の呼出しは通信により媒介するよう変更する必要がある。またプロセス間通信に共有メモリを使用している場合には、分散環境で使用できる共有メモリは殆ど普及していないため、メッセージ通信や共有ファイルによる通信で代替する等、なんらかの変更が必要になる。

これらの問題点に対して、ソフトン方式では次のように対応する。

（1）実際の実行単位であるプロセスではなく、仮想的な実行単位の境界と、具体的な稼働マシンの間の物理境界とを分けて考え、両者の和により実際の実行単位間の実境界が生じると考える。

（2）通信手段は、マシン間や異機種間でも媒介可能な汎用性あるインタフェースを提供する。

（3）具体的な連絡相手の特定や、マシン間での連絡媒介を行うコネクタ部は、本体とは別個の仕様から生成する。

前述の仮想的な実行単位をサブ・プログラムと呼ぶ。サブ・プログラムは、従来の本体部と、本体部間の連絡を行うコネクタ部とからなるソフトンから構成される。分散可能な単位はサブ・プログラムではなくソフトンである。本体部間の連絡は、単純性と汎用性の観点から、サブ・プログラム間の場合はメッセージ通信、サブ・プログラム内の場合は手続き/関数呼出しが良いと判断した。図3にソフトンによるプログラム構成の概念図を示す。

ソフトンの本体部は、従来のプログラミング言語と標準化した部品間インタフェースとを用いて記述する。手続き呼出しは、プログラミング言語の手続き呼出し構文をそのまま使用して記述する。一方メッセージ通信は、各プログラミング言語毎にあらかじめ設定した通信インタフェース手続きを呼び出す形式で記述する。一方ソフトンのコネクタ部は、各ソフトンのインタフェースやソフトンの組合せ、各ソフトンの稼働マシン等の仕様として記述する。こ

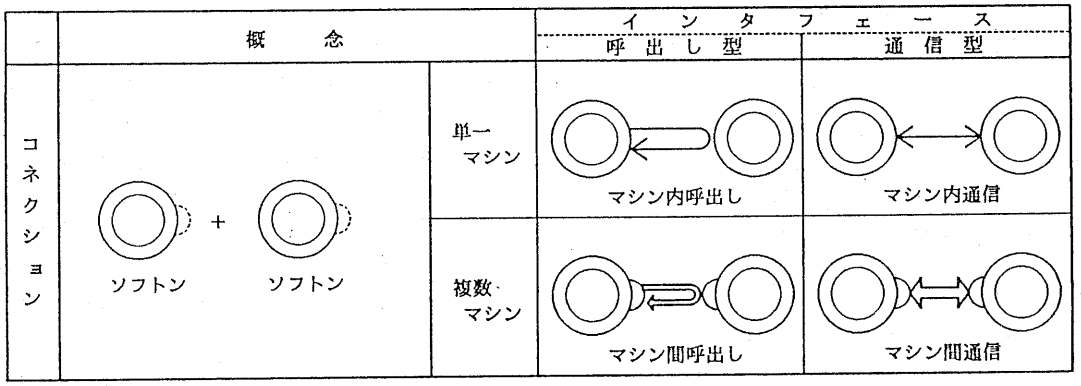


図4　ソフトンとそのコネクション

れらの仕様は専用のコネクタ記述言語を用いて記述する。この仕様に基づいてコネクタ部を生成する。これにより各ソフトンの本体部は、具体的な結合相手ソフトンやソフトンのマシン配置に関わりなく作成することができる。従って本体部とは分離された仕様の変更により、ソフトンの結合相手を簡単に変更できるため、ソフトンの再利用、ソフトウェアの再構成が容易に行える。これを我々はソフトウェアの柔構成方式と呼んでいる。

図4にソフトンとそのコネクションの概念図を示す。本体部はソフトンが単一マシン上に配置されるか複数マシン上に分配されるかに関わりなく同一である。マシン内またはマシン間の具体的なコネクタ・プログラムは、ソフトン・プロセッサ(後述)が生成する。これにより複数計算機を結合した分散環境においても、各計算機上で稼働するソフトウェア(ソフトン)を簡単に結合できる。

4．ソフトン方式

ソフトン方式によるソフトウェアは、図5に示すソフトン・プロセッサにより実現する。ソフトン・プロセッサはコネクタ生成部とコネクション決定部からなる。

(1) コネクタ生成部

コネクタ生成部は与えられたコネクタ仕様に従ってソフトンのコネクタ・プログラムを生成する。この段階では、各ソフトンの稼働マシンや結合相手となるソフトンはまだ特定されていない。従ってコネクタ生成部は計算機内および計算機間のメッセージ通信や手続き呼出しを媒介するコネクタ・プログラムをソフトン毎に作成する。

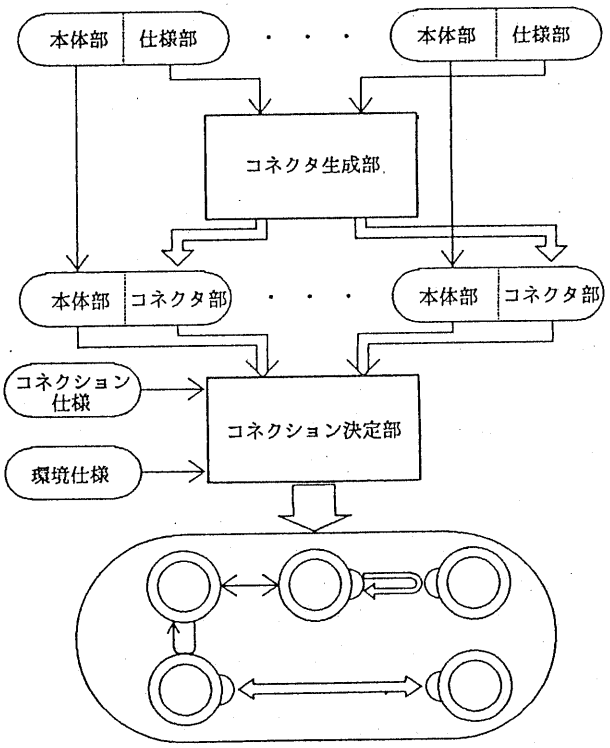


図5　ソフトン・プロセッサのシステム構成

(2) コネクション決定部

コネクション決定部は、各ソフトンの稼働マシンを特定する稼働環境仕様、およびコネクタの結合相手を特定するコネクション仕様に基づき、適切なコネクタ・プログラムを取捨選択し、ソフトンを結合する。

５．ソフトンの類型化

ソフトウェアの機能部品化(ソフトン化)を実施するにあたっては、何を一つの機能部品ととらえるかがソフトウェア設計上の最重要課題である。これの善し悪しにより、ソフトウェアの開発生産性、保守生産性が大きく左右されると共に、その機能部品がソフトンとして有用かどうか、即ちそのソフトンが再利用や再構成に値するか否かが決まってくる。そこでソフトン化の指針としてソフトンの類型を定めることにした。

５．１　基本方針

ソフトウェアのソフトン化を行うための基本的考え方は、

（１）ソフトンの結合機構を明確化、限定化する

（２）問題の種別に応じたソフトンの類型(機能/インタフェース)を制定する

ことである。これにより、

（ｉ）ソフトンの組合せに関して、形式上の障害を排除し、（ii）限られた類型に準拠することにより、設計の過度の自由度を制限し、（iii）限られた類型に準拠することにより、ソフトンの機能の明確化、単純化を図る、の効果を狙っている。

５．２　結合機構

ソフトン間の結合はメッセージ通信と手続き/関数呼出しであるが、そのコネクタ形式はさらに次のように細分される。

（１）送信/受信専用通信コネクタ

メッセージ送信または受信専用の通信コネクタ。送信コネクタは受信コネクタとのみ接続できる。

（２）送受信共用通信コネクタ

メッセージの送信と受信の両方が可能な通信コネクタ。送受信コネクタは送受信コネクタとのみ接続できる。

（３）呼出しコネクタ

手続きまたは関数の呼出しを行うコネクタ。

図６に通信コネクタおよび呼出しコネクタの概念図を示す。

５．３　類型化の検討事項

類型を定めるにあたっては、ソフトンがどのように使われる(べき)か、どのような機能を持つ(べき)かを配慮する必要がある。ソフトンの機能設定における基本的考え方は以下のものである。

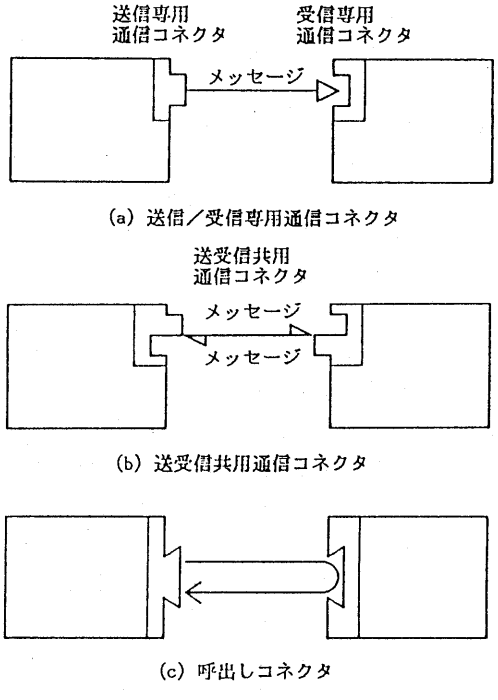


図６　コネクタの概念図

（１）部品は単機能である

（２）機能は部分機能に分割されて階層構造(hierarchy)を持つ

（３）部品は(できるだけ)他の部品と独立した閉じた世界を構築する

（４）ソフトウェアは機能のレベルによって層構成(layer)をとる

ソフトンの類型を設定するには、ソフトン間の関係に関して以下の事項を考慮する必要がある。

（１）相互関係

ａ）依　頼：一方からの依頼によって他方が稼働し、依頼元の要求する機能を果たす。

ｂ）分　業：一連の処理を分割した個々の処理を施し、全体としてまとまった機能を果たす。

ｃ）指　令：独立した機能の実行を指示し、これに応答する。

（２）層構成

ａ）上　下：異なる層に属するソフトン間のインタ

ラクション。

b）左　右：同一の層に属するソフトン間のインタラクション。

（3）実行制御

a）同　期：実行を中断し、インタラクションした相手ソフトンの応答を待つ。

b）非同期：実行を中断せず、インタラクションした相手ソフトンの応答を待たない。

（4）相互作用

a）依　存：インタラクションした相手ソフトンの実行結果を直接必要とする。

b）非依存：インタラクションした相手ソフトンの実行結果は直接必要とせず、相手ソフトンの実行効果を間接的に利用する。

（5）結合手段

a）通　信：メッセージの伝達により相手ソフトンとインタラクションする。

b）呼出し：手続きまたは関数呼出しにより相手ソフトンとインタラクションする。

（6）生死／活動

a）生成／消滅：インタラクションする毎にソフトンが生成／消滅する。

b）起動／停止：ソフトンは事前に存在し、インタラクションする毎に起動/停止する。

5.4　ソフトンの型

従来のモジュールは、モジュール間のインタフェースの設定も含んでいる。一方ソフトンは、機能部品を処理の本体部とコネクタ部とに分け、コネクタの可変性を実現するものである。そこで5.3で提示した検討事項のうち、ソフトン間の関係に着目する「相互関係」をソフトン類別化の基本とし、依頼、分業、指令関係をそれぞれサーバ/クライアント型、フィルタ型、プロセス型と名付けることにした。各型の概要と使用可能なコネクタは下記のようになる。

（1）サーバ/クライアント型

他のソフトンからの依頼を受け機能を果たした後、依頼元へ結果を返すソフトン。依頼元ソフトンは同一層内または上位層のクライアント型ソフトンでなければならない。サブルーチン、サブシステムやウィンドウ、ネットワークとのインタラクション等はこのサーバ/クライアント型である。サーバ/クライアント型ソフトンは次のコネクタを使用する。

・呼出しコネクタ

・送受信共用通信コネクタ

（2）フィルタ型

他のソフトンからデータを受け取り加工を施した後、結果を他のソフトンに送り出すソフトン。このうち送り出しのみを行うものをソース型、受け取りのみを行うものをシンク型と呼ぶ。受け取る相手はフィルタ型またはソース型、送り出す相手はフィルタ型またはシンク型で、いずれも同一層内のソフトンでなければならない。フィルタ型ソフトンは次のコネクタを使用する。

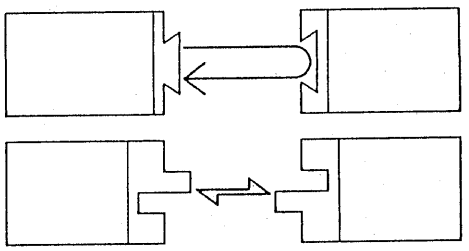
・送信専用通信コネクタ

・受信専用通信コネクタ

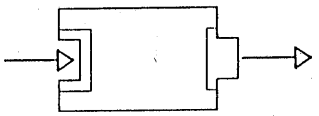
（3）プロセス型

他のソフトンからの指令に基づき機能を実行するソフトン。相手のソフトンは、同一層内、または隣接層内のソフトンである。プロセス型ソフトンは次のコネクタを使用する。

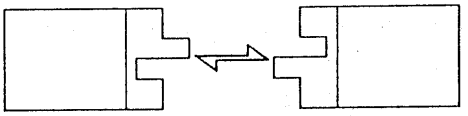
・送受信共用通信コネクタ



(a) サーバ/クライアント型



(b) フィルタ型



(c) プロセス型

図7　ソフトンの類型

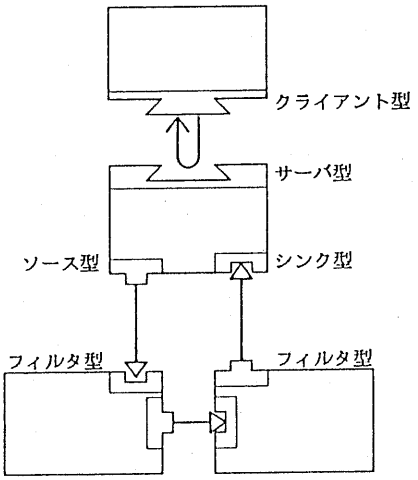


図８　ソフトンと類型の対応

図７にソフトンの類型別の使用可能なコネクタを示す。ところで、１つのソフトンは複数の類型にあてはまることがある。例えば、あるサーバ型ソフトンが、他のフィルタ型ソフトンを使用してその機能を実現する場合には、クライアント型ソフトンに対してはサーバ型であるが、フィルタ型ソフトンに対してはソース型、あるいはシンク型となる(図８参照)。このようにソフトンと類型との対応は１対１ではないが、対応する型のインタフェース間の結合のみが許される。

６．例　　題

ここでは文献[7]に示されている会話型システムに若干の機能拡張を施した会話型エディタのソフトンによる実現例を示す。

本エディタは、ソース・ファイルの編集機能と、インタプリタによるプログラム実行機能、プログラムのコンパイル／リンク機能を持つ。このエディタを次の９個のソフトンから構成する。

１）edit：エディタのメイン部
２）run　：インタプリタ
３）file：ソース・ファイル管理
４）kbd　：キーボード入力
５）dsp　：画面出力
６）comp：コンパイル／リンクの実行制御
７）prp　：プリプロセッサ
８）cmp　：コンパイラ
９）link：リンカ

このうち(1)、(2)は本エディタ固有の層であり、(3)～(9)は他のアプリケーションからも利用可能な共通の層であると考えられる。また(2)～(6)、(9)は、サーバ的な機能である。但し(6)、(9)は、エディタとは独立した機能と考えられる。従って(2)～(5)は呼出しコネクタを用いるサーバ型とし、(6)、(9)は送受信コネクタを用いるプロセス型とする。一方(7)、(8)はソース・プログラムをオブジェクト・プログラムに変換する一連の機能であるから、フィルタ型が適当と考えられる。

この結果、各ソフトンのコネクタ仕様は図９に示すものとなる。同図において、((…)) は呼出される側の呼出しコネクタ、(…) は呼出す側の呼出しコネクタ、{{…}} は受信コネクタ、{…} は送信コネクタ、[…] は送受信コネクタをそれぞれ意味する。図１０にコネクション仕様の例を、図１１にこの時のソフトン構成をそれぞれを示す。

```
edit : (get,put,run,kbin,disp)  [comp]
run  : ((run))  (get,disp)
file : ((get, put))
kbd  : ((kbin))
dsp  : ((disp))
comp : [comp,link]  (start)  {(end)}
prp  : {(start)}  (end)  (get,put)
cmp  : {(start)}  (end)  (get,put)
link : [link]  (get,put)
```

図９　コネクタ仕様

```
edit : (get,put)      =file : ((get,put))
edit : (run)          =run  : ((run))
edit : (kbin)         =kbd  : ((kbin))
edit : (disp)         =dsp  : ((disp))
edit : [comp]         =comp : [comp]
run  : (get,put)      =file : ((get,put))
comp : [link]         =link : [link]
comp : {start}        =prp  : {(start)}
prp  : (end)          =cmp  : {(start)}
prp  : (get,put)      =file : ((get,put))
cmp  : (end)          =comp : {(end)}
cmp  : (get,put)      =file : ((get,put))
link : (get,put)      =file : ((get,put))
```

図１０　コネクション仕様

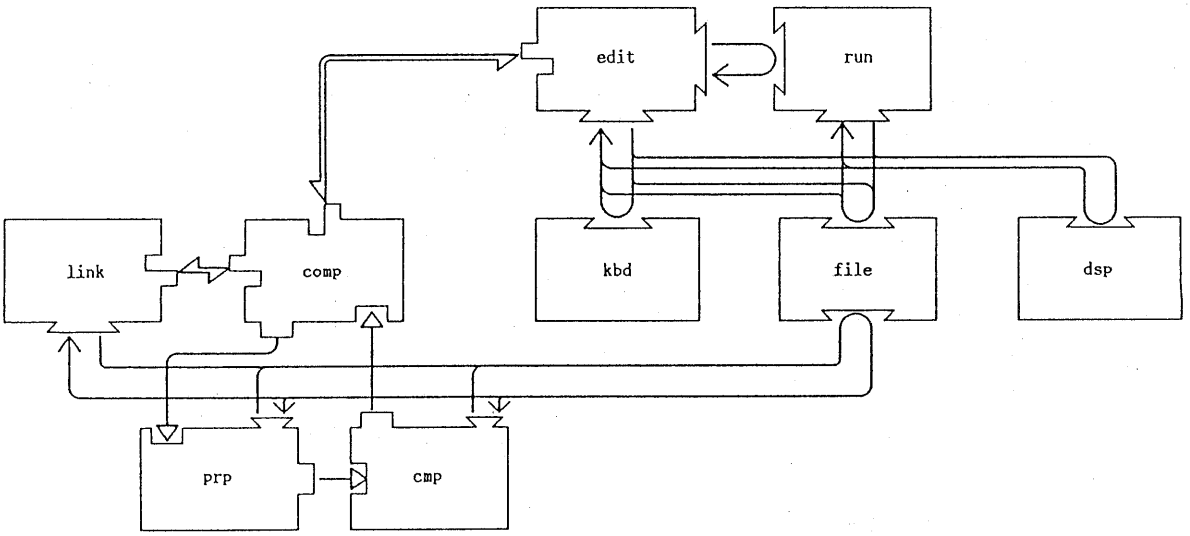


図１１　会話型エディタのソフトン構成

７．む　す　び

ソフトウェアの機能部品の再利用、再構成と、分散環境への適応を容易化するソフトウェア柔構成方式として、ソフトンという部品概念とソフトンによるソフトウェア構成を提案した。従来のトップダウンな機能分割方式は、確定した問題を一度に完全に解く場合には適しているが、モジュールを部品として使用する際の変更や再構成にはあまり適していない[8]。一方オブジェクト指向方式は、コンポーネントの積み上げ方式であるため変更には強い。しかしながらインヘリタンスあるいはデリゲーション機構[9]によるオブジェクト間の階層構成を前提としているため、各オブジェクトを部品として使用する場合には、その上位にある暗黙の階層構成を意識する必要がある。これに対してソフトン方式では、部品間の結合部を部品本体部とは切り離し、かつ各部品は暗黙の階層を持たない平坦なものである点に特徴がある。

今後の課題としては、異機種(OS)、異言語間での協調実現方式、ソフトンの動的な結合、動的な生死・活動・移動の操作方法の検討等がある。

参　考　文　献

1) B.Cox : Object-Oriented Programming－An Evolutionary Approach : Addison-Wesley, 1988

2) R.Raj, H.Levy : A Compositional Model for Software Reuse : The Computer Journal Vol.32 No4, '89

3) A.Birrell : Implementing Remote Procedure Call : ACM Trans. Computer Systems Vol.2 No1, Feb.84

4) M.Jones, R.Rashid, M.Tompson : Matchmaker:An Interface Specification Language for Distributed Processing : Proc. 12th ACM SIGACT/SIGPLAN Symp. Principles of Programming Language, Jan.85

5) M.Jones, R.Rashid : Mach and Matchmaker:Kernel and Language Support for Object-Oriented Distributed System : Proc. OOPSLA '86

6) P.Gibbons : A Stub Generator for Multilanguage RPC in Heterogeneous Environment : IEEE Trans. Softw. Eng. Vol.SE-13 No1, Jan.87

7) P.Henderson : Purely Functional Operating Systems : Functional Programming and its Applications, Cambridge Univ. Press 1982

8) B.Meyer : Reusability:The Case for Object-Oriented Design : IEEE Software, Mar.87

9) H.Lieberman : Using Prototypical Objects to Implement Shared Behavior in Object Oriented Systems : Proc. ACM Conf. Object-Oriented Programming Systems, Language and Applications, 1986