Concerns Oriented Programming for Collaborative Learning

武田　俊之 [1)]
Takeda Toshiyuki

1) E-mail: takeda@kwansei.ac.jp ）

ABSTRACT. This paper describes a new technique and interface named Concerns Oriented Modeling to design cognitive user models and distributed application architecture for computer supported collaborative learning(CSCL). It is based Linda, coordination model for distributed network systems.

1　背景及び目的

CSCL (Computer Supported Collaborative Learning) の分野は著しい進歩を見せている。しかし、システムの設計については問題点がある。それは、たとえば Latency、Synchronization、Partial Failure といった分散ネットワークシステムの本質的な欠点である。これを回避するために、CSCL システムは様々な工夫をしているが、それがかえってデータやオブジェクトの再利用性を損なったり、システム連携においては performance、scalability、resource sharing、fault torelance、elegance といった分散ネットワークシステムの利点を損なう場合もある。

このプロジェクトでは、分散ネットワークシステムとしてよく利用される Remote Procedure Call に代わって分散協調モデルの Linda を採用し、さらにオブジェクト指向のクラス階層を拡張する Environmental Acquisition を Linda と組み合わせることによって、安全で理解しやすいシステムを構築できるような基本システムの開発を目的とする。これを Concerns Oriented Programming と呼ぶこととする。小さく、プログラマだけではなく利用者にとっても理解しやすい設計の単位として、関心（Concern) を用いるためである。

2　分散システムモデルにおける関心主導モデル

分散アプリケーションは複数のコンピューティング・デバイスからなるネットワーク上に遍在するプロセスの集合と、それらのプロセスが協同して動作することによって、タスクを遂行するようなアプリケーションである。分散ネットワーク・システムは分散アプリケーションの基礎となる。

協調作業をコンピューターによって支援するために、分散ネットワーク・システム上にアプリケーションを構築することは自然なことである。なぜならある時間内に同じ資源を読み、書き、更新することが協調作業だからである。まず本稿でモデル化する分散システムについて定義しよう。ネットワークで接続された複数のマシン上に存在する複数のプロセスが協調しながらオブジェクト（オブジェクト指向言語でいうところのオブジェクト）を共有、操作するシステムとする。分散システムを設計する上で意識しなければならないのはマシン境界とプロセス境界である。

Freeman [3] らは以下の 5 つの点を分散コンピューティングの利点として挙げている。

1) Performance: タスクを分割することのできる問題であれば、分割したタスクを複数のプロセッサに割り振ることによって総実行時間を削減することができる。

2) Scalability: 単一マシンでは CPU やメモリ・サイズといった性能の限界に直面するような場合でも、適当にタスクを分割して実行する分散システムでは総タスクのサイズに応じてマシン数を増やせばよい。

3) Resource sharing: 高価な計算資源やデバイス、膨大なデータを共有することができる。

4) Fault tolerance and availability: 分散アプリケーションは一部のプロセスやコンピューティング・デバイスに障害があっても他のプロセスやデバイスが協調して動作を続けることができる。

5) Elegance: プロセスが並行動作する分散アプリケーションとして設計することが自然なタスクが存在する。

分散システムを実装するためのモデルとしてよく用いられるのは、メッセージ・パッシングとリモート・プロシージャ・コール (RPC) である。メッセージ・パッシングではプロセス間でメッセージをやりとりする。RPC では、遠隔オブジェクトのメソッド呼び出しを遠隔オブジェクトのインターフェースを持ったスタブ・オブジェクト経由で起動する。

Freeman は分散コンピューティングの困難さについても以下の 3 点を挙げている。

1) Latency: 複数のプロセスがマシン境界を越えてネットワーク経由で通信する場合、遅延 (latency) が発生することがある。遅延は誤動作の原因になることもある。

2) Synchronization: プロセスが同期をとって動作するよう設計するのは容易ではない。たとえば、共有データを更新する場合、同時に複数のプロセスが更新しないような工夫が必要である。多くの場合、このためのコントロール・オブジェクトを用意する。

3) Partial failure: 分散システムにおいてその一部 (プロセスやプロセッサなど) 障害が発生した場合、障害とその理由を検知した上で、アプリケーションの実行を正しく継続することは容易ではない。

Suthers [5] は CSCL アプリケーション Belvedere のシステム・デザインの進化を振り返りながら、協調学習環境のアーキテクチャについて考察している。

1) MVC (Model-Iview-Controller) モデル (図 1-a): オブジェクトの内容 (model) を見かけ (view)、操作 (controller) と分離することによって、設計の見通しをよくし、メンテナンス性を高めるオブジェクト指向における有力な設計パターン。

2) Centralized architecture (図 1-b): MVC のうち Control と View だけ (view はしばしば表示用の結果だけ) を転送する。ネットワークのバンド幅が必要なこと、ネットワークに障害があるとき、まったく利用できなくなることが欠点である。

3) Replicated architecture (図 1-c):コントローラー・プロセス同士が通信する。必要なネットワークのバンド幅は少ない一方で、model の一貫性を維持することがむずかしい。

4) Distributed achitecture (図 1-d):サーバー上にあるモデルをクライアントの View / Controller が操作する。必要なネットワークのバンド幅も少なく、model の一貫性も保ちやすいが、ネットワークに障害があるとき、まったく利用できなくなることが欠点である。

5) Hybrid architecture (図 1-e): Replicated と Distributed の利点を組み合わせたもので、クライアント・マシンにもモデル（のコピー）を配置する。ネットワークに障害があっても利用することができるが、一方複数のマシンがモデルを更新してデータの一貫性を失わないよう制御する必要がある。

Suthers の考察からもわかるとおり、メッセージ・パッシングや RPC を使った分散システムではどこかに一元管理をおこなうデータのマネージャー・プロセス（またはルーチン）が存在する。なぜならデータは常に整合性を維持されなければならなず、マネージャーがなければ複数箇所でデータが更新されたときにシステム内でのデータの一貫性が維持できなくなるからである。マネージャー・プロセスはマシン境界またはプロセス境界のところに存在するのが普通である。複数のプロセスが同時にデータ構造の部分を更新すると、全体が破壊されることもありえる。

2.1 Linda モデル

メッセージ・パッシングとリモート・プロシージャ・コールの欠点を克服するために分散オブジェクト構造とその操作言語のモデルである Linda ([4] , [2]) が考案された。Linda はタプル空間 (tuple space) と呼ばれる複数のプロセスで共有されているデータのコンテナを用いて、プロセス間通信でデータ共有をサポートする分散並列プログラムのモデルである。タプル空間は一種の共有メモリであるといえる。 Linda は C 言語や Lisp、Prolog など普通のプログラミング言語 (Linda ではこれを computing language と呼ぶ) に実装され、並列プログラミングのためのタプル・スペース・モデル (tuple space model) を組み込んだいくつかの簡単な命令から構成されている。この命令群を Linda では coordination language という。 C-Linda を例にとって Linda モデルとその典型的な実装を説明する。C-Linda には、4 つの基本タプル命令 out、in、rd、eval、2 つの変形命令 inp と rdp がある。(inp と rdp は説明を省略。)

- out(t) はタプル t をタプル空間に追加する。これを実行したプロセスは実行を継続する
- in(s) は問い合わせタプル s に一致するタプルをタプル空間から取り出す。問い合わせタプルはデータの列であり、数値や文字列のような値を持つか、空間内のタプルとマッチするような変数である。C-Linda では変数の前に疑問符をつける。
- rd(s) はマッチしたタプルが空間から取り出されるのではなく、コピーされる点が in(s) と異なる。
- eval(t) は、t がタプル空間に追加される前に評価される点が out(t) と異なる。

メッセージ・パッシングや RPC とは異なり、Linda のプロセスが空間とやりとりするのはオブジェクトだけであって、メッセージを受信する宛先は指定されない。任意のプロセスが必要に応じて空間からデータを取りだして処理する。また、原則として送信元はセットされないので、処理された結果はデータを送り出したオブジェクトに返されるわけではなく、単に空間に書き込まれるだけである。また、Linda モデルで用いる協調 (Coordination) のプリミティブは、基本的に in、out、rd だけで非常に単純である。

Linda で用いられる分散データ構造でもっとも特徴的なのは配列などの構造をもったデータの表現であろう。Linda モデルでは複数のプロセスが確実に安全に配列の要素を更新できるように配慮されている。たとえば、[10, 9, 8] という配列は、仮にこれに a という名前をつけたとして、(“ a ”,1, 10)、(“ a ”, 2,9)、(“ a ”, 3, 8) の三つのタプルと

して表現される。タプルは、名前、配列番号、値の列として表現されている。実際にタプル空間でこれを利用するにはさらに二つのタプル、(“ a ”, “ start ”, 1)、(“ a ”,“ end ”, 3) も追加する。それぞれ、配列の先頭と末部の添え字を表現している。これが必要な理由は、タプル空間から配列の要素タプルが取り出されているときに、先頭と末尾が何かわからなくなる可能性があるからである。図に示したように配列の要素は独立に更新することができる。　以上のように Linda モデルの分散データ構造は、複数のプロセスが同時に安全にデータ構造へのアクセスと変更を可能にするすぐれたモデルである。しかし、Linda モデルには Bacon [1] の指摘するような以下の欠点も存在する。

- タプルを区別するのに、簡単な構造化されていないストリングの形での名前つけを使用している
- すべてのプロセスがタプル空間を共有する場合、保護機能が潜在的な問題点である。どのプロセスでも、整合する任意のタプルを読み出しまたは取り出すことができる。これは小規模なアプリケーションでは容認されるが、必ずしも互いに信用できない一般的なシステムのプログラミングでは問題である。
- 集中システムでさえも、効率的なインプリメンテーションを達成するのが困難である。インプリメントするためには、メッセージシステムと同様に、単にメッセージのヘッダだけでなく、タプルの種々のフィールドの内容を知っている必要がある。
- 共有されたグローバルなタプル空間の抽象化を、分散して有無でどのようにしてインプリメントできるかが明らかではない。

Linda は、あるタスクを分散的に解決するアプリケーションのためのモデルであるので Bacon の指摘はそのまま当てはまらない。しかし、システムとして利用する資源、オブジェクトの再利用という観点からすると、Linda モデルを多様なアプリケーションが同時に動作することのシステムに適応できるよう拡張することは意義のあると考えられる。特に利用者が同じ資料を用いてさまざまな観点から吟味、作業することが必要となる協調学習を支援するシステムのためには重要であろう。

2.2 Linda モデルの拡張としての関心主導モデル

Linda モデルの弱点を克服するために、関心主導モデルを導入する。このモデルでいう「関心」とは Linda モデルの分散データ構造ではバッグに相当する。その要素はインデックスもキーも持たない。配列や辞書に相当するデータ構造は、関心の属性を継承して作成する。

関心は空間ではないが、それへアクセスするオブジェクトから見ると空間からかわらないインターフェースをもつ。すなわち、データを追加する out、データを取り出す in、データをコピーする rd の 3 つのインターフェースを持つ。関心へオブジェクトが追加されたとき、関心はそのオブジェクトの型によって決まるラッパーオブジェクトを生成する。ラッパーオブジェクトへメッセージが送られた場合、内容オブジェクトへのメッセージを委譲する。

もし、その関心自身が別の関心の要素としてラッパーオブジェクトにくるまれている場合、上位の関心にオブジェクトが out される。これは終端の関心に到達するまで続く。

関心オブジェクトへメッセージが送られたにもかかわらず、その処理方法がわからない場合、関心内の要素名が検索され、もしその名前のオブジェクトがあった場合はその要素を返す。これによって、名前空間を柔軟に設計することができる。

関心主導モデルでのデータはただのタプルではなく属性リストとして表現される。タプルは数字や文字列の列なので、その意味するところを知っているプロセスでなければ処理をすることができない。しかし、属性リストであれば、属性名をオブジェクト指向言語で使われるインターフェースのように用いることができる。

関心主導モデルのオブジェクトは空間内でユニークなオーナー ID とオーナーの所有するオブジェクト内でユニークな ID を持つ。オーナー ID と ID の組み合わせによって、空間内でオブジェクトを特定できる。オーナー ID は一種の名前空間としても利用可能である。すなわち、あるオーナー ID を持つオブジェクトの集合は、そのオーナー ID を持つ人の関心すべての集合である。

3 開発内容

3.1 基本クラスライブラリ

分散環境において適当な粒度で分割、管理されているデータ構造と、それを非同期に操作するための言語モデル、Linda をベースとして基本クラスライブラリを Python 言語および Smalltalk 言語で開発した。Linda を利用するとデータの更新がかならずアトミックにおこなわれるために非常に安全な分散システムが構築できる。

一方既存の Linda はアプリケーションを分散化するために設計されているために、プログラムやデータの再利用性が低いという欠点がある。データは Linda で用いられる Tuple ではなく、連想リストを用いた。また、グローバルユニークとなる Owner と ID の属性を持たせることによって、複数システム間でデータの再利用が可能となっている。

また、Linda への拡張として Environmental Acquisition というコンセプトを導入した。これはオブジェクトへのセレクタを解釈するときに、環境への検索もおこなうというものである。

基本オブジェクトは owner、id の他に作成、更新、アクセスの時刻を持つ。

外部システムとのインターフェースのために、

XML からの変換ライブラリも開発した。XML 化したオブジェクトの例を以下に示す。このシステムのオブジェクトは単なる連想リストであるために可読性が非常に高い。

```
<note>
    <owner>01002003</owner>
    <id>93484884</id>
    <creationdate>Dec 25, 2002</creation date>
    <modifieddate>Dec 25, 2002</modifieddate>
    <accessdate>Dec 25, 2002</accessdate>
    <author>Minnie</author>
    <title>What is his name?</title>
    <content>What is his name?

    He is so cute!
    </content>
    <version>0</version>
</note>
```

Linda モデルに従って非同期かつアトミックに共有レポジトリを操作することによって安全性が高まった。また、Environmental Acqisition によって非常に見通しのよいプログラムが書けるようにになったと開発者は考えている。

3.2 レポジトリとアクセス API

レポジトリには、MySQL をデータベースとして採用した。アクセスライブラリは Python で記述、オブジェクトと RDB のマッピングをここで記述している。

外部のアクセス用に Web サーバーの CGI として動作する入出力インターフェースを開発した。データ交換のために XML を用いている。

現在のバージョンでは同じ量のデータを RDB に格納するよりも確実に遅い。（未計測）これは、Python と MySQL のインターフェースが低速であることと、オブジェクト-RDB のマッピングが最適化されていないことが理由であると思われる。設計を最適化するか、XML ネイティブサポートのあるデータベースを採用することによって速度を向上させることが今後の課題である。

3.3 CMFPink

Pink [6] は Web アプリケーションサーバー Zope 上に実装されたアプリケーションで、文書やプログラムコード、Web ページへのリンクに対して注釈をつけることができ、その注釈についての議論の環境を提供することによって、参加者間の意識を発散させずに議論を維持することができるようなシステムである。(Figure 1 参照)

この Pink のドキュメント（のフラグメント）や議論の一つ一つを上で記述したリポジトリに格納するように拡張した。システムのイメージを Figure 2 に示す。

関心主導による拡張によって、Pink は他のアプリケーションとデータオブジェクトを共有するこ

図 1: Pink: document with showing discussion

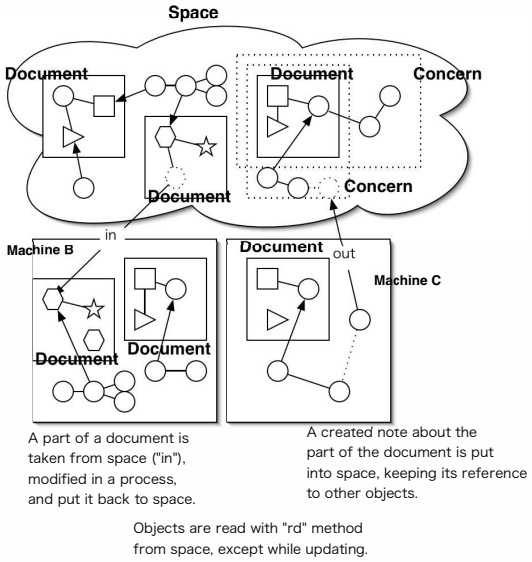

図 2: Pink software architecture with concerns-oriented model

とができるようになった。CMFPink は環境が整い次第オープンソース・ソフトウェアとしてリリースされる。

3.4 Annotation Enabled System-Browser

プログラムのソースコードを読むことはプログラマの学習において非常に重要であるが、コードに疑問点があったとしても一人で解決することはむずかしく、また、近くにメンターとなる人がいない場合にはそのまま理解せずにおわってしまうことも多いと思われる。

Smalltalk (Squeak) の SystemBrowser に、注釈とコメントを入力するインターフェースを追加した。新しくつけられた注釈やコメントはレポジトリを介して他の人のイメージ上に伝搬されるように設計、実装されている。

Annotation Enabled SystemBrowser は環境が整い次第オープンソース・ソフトウェアとしてリリースされる予定である。（Figure 3）

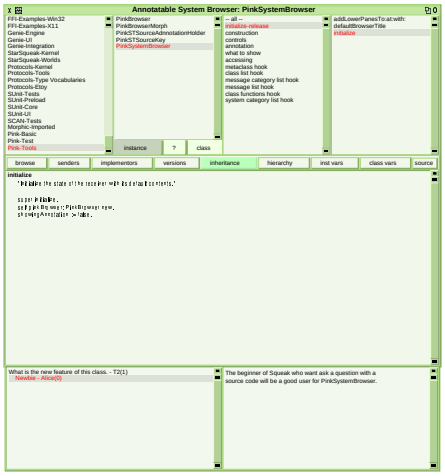

図 3: PinkSystemBrowser

3.5 Annotation Enabled Morph

Squeak プログラミングは GUI を用いたオブジェクトの作成とそれらを操作するスクリプティングからなっている。疑問点があった場合にメモを付箋として貼るとそれがレポジトリを介して他の人のイメージに伝搬するようなシステムを設計、実装した。(Figure 4)

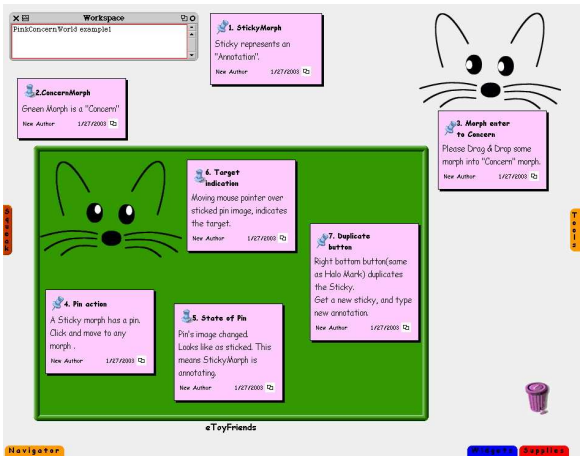

図 4: Annotation Enabled Morph

Annotation Enabled Morph は環境が整い次第オープンソース・ソフトウェアとしてリリースされる予定である。(Figure 4)

4 今後の課題、展望

情報機器のモビリティへの需要性が高まることで、今回のプロジェクトの成果の重要性も生かされると考えられる。また、P2P アプリケーションの基本システムとして活用することもあると思われる。

最も大きな課題はレポジトリの速度である。今回は分散レポジトリの設計とテストができなかったが、レポジトリの分散化は高速化に役立つであろう。また、オブジェクトと RDB のマッピングもアクセス API から正確に動作するかどうかを重要視したため、効率についてはほとんど考慮していない。プロジェクト中に若干の検討を開始したが時間切れとなった。これは今後の課題としたい。

5 まとめ

協調分散学習のモデリングから、システムの設計、実装にまで使えるものを目指して関心主導というコンセプトを導入、それにもとづいたシステムを試作した。レポジトリとして Linda 的なモデルを用いて一部が動作しなくても安定して動作することができることを目的とする。レポジトリとアクセスライブラリは MySQL と Python で実装した。

応用システムとして、(1) 協同である文書について議論するためのシステム CMFPink を Web アプリケーションサーバー Zope に、(2) ソースコードにコメントを入れられる Annotation Enabled System Browser と (3) 開発しているオブジェクトについてコメントをやりとりすることのできる Annotation Enabled Morph を Squeak 上に実装した。

これらのソフトウェアは環境が整い次第公開される予定である。

6 Acknowledgements

プロジェクト・マネージャーとして導いてくれた Alan Kay、Kim Rose 両氏、開発、テストで貢献してくれた林徹也氏、横川耕二氏、森巧尚氏、山田陽子氏、プロジェクト管理組織として事務作業をしていただいた山本麻起子氏はじめ ASTEM の方々に感謝をいたします。

参考文献

[1] J. Bacon. Concurrent Systems. Addison-Wesley Publishing, 1993.

[2] N. Carriero and D. Gelernter. How to write parallel programs. The MIT Press, 1990.

[3] E. Freeman, S. Hupfer, and K. Arnold. JavaSpaces Princicles, Patterns, and Practice. Addison Wesley, 1999.

[4] D. Gelernter. enerative communication in linda. ACM Transactions on Programming Languages and Systems, pages pp.80–112, January 1985.

[5] D. Suthers. Architectures for computer supported collaborative learning. In IEEE International Conference on Advanced Learning Technologies (ICALT 2001), 6-8 August 2001.

[6] T. Takeda and D. Suthers. Online workspaces for annotation and discussion of documents. In Kinshuk, R.Lewis, K. Akahori, R. Kemp, T.Okamoto, L. Henderson, and C.-H. Lee, editors, Proceedings of the International Conference on Computers in Education (ICCE2002), pages 1294–1298. IEEE Computer Society, December 3-6 2002.