論　文

ソフトウェア開発におけるインタラクションの支援方式

松下　誠[†]　飯田　元[††]　井上　克郎[†]

An Interaction Support Mechanism in Software Development

Makoto MATSUSHITA[†], Hajimu IIDA[††], and Katsuro INOUE[†]

あらまし　本論文では，ソフトウェア開発プロセスにおける各要素間のインタラクションに着目したモデル化手法と，本モデル化に基づいたソフトウェア開発環境の提案を行う．本手法では，ソフトウェア開発時におけるインタラクションはエージェントと通信チャネルの集合としてモデル化される．エージェントは通信チャネルを用いて他のエージェントとの情報交換を行う．通信チャネルは通信する内容はその形態に応じてクラス分けされる．我々はまた，本手法を用いたソフトウェア開発支援環境の試作を行った．試作した環境はエージェントのための代理プログラムと統合された通信サーバからなり，インタラクションをはじめ，プロセスの実行やユーザの作業誘導を行う．

キーワード　ソフトウェアプロセス，インタラクション，ソフトウェア開発支援環境

1. まえがき

近年のソフトウェア開発はより分散化してきており，ネットワークを用いた開発形態はごく一般的になってきている[1]．このような開発形態においては，従来の1箇所に集中して行っていた開発とは異なり，進ちょく状況を開発者が相互に理解し，かつ開発者間の相互連絡を確立することが非常に重要となる．

これらの問題を解決するために，ソフトウェア開発におけるそれぞれの開発作業を明らかにし，それら相互の関連を考えるソフトウェアプロセスの研究が行われている[2],[10],[13]．これらの研究では，ソフトウェア開発作業とそれに関連する開発環境を再構成することによって問題の解決を図っている．ソフトウェアプロセスに関する研究分野として，ソフトウェアプロセスをモデル化して記述するプロセスプログラミング[11]や，記述されたソフトウェアプロセスの定義に基づいてソフトウェア開発の作業者を支援するプロセス中心型開発環境の構築[3],[4],[12]などが挙げられる．

[†]大阪大学基礎工学研究科，豊中市
Graduate School of Engineering Science, Osaka University, 1-3 Machikaneyama-cho, Toyonaka-shi, 560-8531 Japan
[††]奈良先端科学技術大学院大学情報科学センター，奈良県
Information Technology Center, Nara Institute of Science and Technology, 8916-5 Takayama, Ikoma-shi, 630-0101 Japan

ソフトウェア開発作業を，作業者間の対話や中間生成物の受渡しといったインタラクションに着目してとらえた場合，ソフトウェア開発作業自体はさまざまなインタラクションから構成されると言える．各開発者は，さまざまな対象とインタラクションを行いながら開発作業を進行させている．しかし，従来提案されていたソフトウェアプロセスモデルやそれに基づいたプロセス中心型開発環境では，これらのさまざまな種類のインタラクションは明示的に扱われていないか，若しくは単純な作業の型の一つとしてしか扱われていないため，インタラクションの記述や支援が十分ではない．

本研究では，ソフトウェア開発プロセスを構成する要素間のインタラクションに着目したプロセスモデルを提案する．また，インタラクションを支援する開発環境の設計を行う[8],[9]．本モデルではソフトウェア開発環境におけるインタラクションを実際にインタラクションを行う主体としてのエージェントと，インタラクションの内容を表現する通信チャネルの集合として表す．

以下 **2.** では，ソフトウェアプロセスにおけるインタラクションについて考察する．**3.** では，インタラクションのモデル化方法について説明する．**4.** では，提案するモデルに基づいた開発支援環境について述べる．最後に **5.** で本研究についてまとめる．

2. ソフトウェアプロセスとインタラクション

本章では，ソフトウェアプロセスにおける開発作業とインタラクションについて述べる．

2.1 ソフトウェア開発のためのプロセスによる支援

ソフトウェア開発作業の各要素をソフトウェアプロセスとして記述する方法はさまざまな方法がある．現在の開発作業をソフトウェアプロセスとして記述することによって，開発組織全体で適用する標準プロセスやプロセスの開発環境への適応などが非常に簡単に行える．

また，記述されたソフトウェアプロセスは量的あるいは視覚的方法による進ちょく状況の把握にも用いることができる．形式的に記述されたソフトウェアプロセスを用いることによって，記述自体を計算機によって自動的に実行させることができる．また，ソフトウェア開発のシミュレーションを行うための基礎とすることができる．

2.2 ソフトウェア開発中のインタラクション

ソフトウェア開発環境には，さまざまな種類のインタラクションが存在する．例えば「生成物の受渡し」「作業の開始/終了通知」「共同作業者間での会話」などはインタラクションである．また，インタラクションの内容は多岐にわたる．例えば，インタラクションの例として，次のような状況を想定する（図 1）．

> あるアプリケーションの開発が行われているとする．このアプリケーションは共通ライブラリー（*libR*, *libS*, *libT* の三つ）と，それを使ういくつかのコンポーネントから構成されているとする．
> 共通ライブラリーの開発を行う作業者が 3 名（*A*, *B*, *C*）おり，各ライブラリーはそれぞれ複数人で開発を行っており，開発責任者があらかじめ決定されているとする．ライブラリーの開発責任者はライブラリーの開発状況を把握する仕事を受け持つ．
> また，コンポーネントの開発者，つまり，共通ライブラリーを利用する作業者が 4 名（*X*, *Y*, *Z*, *W*）おり，各開発者は（一つ以上の）共通ライブラリーを利用してコンポーネントの開発を行っている．これらの開発者は，自分が利用する共通ライブラリーに関する告知（例えば仕様変更やバグフィックス報告など）を知りたいと思っており，あるいはライブラリーの開発者に対して質問を行ったり，他のライブラリーの利用者とライブラリーに関する情報交換を行いたいとも思っている．

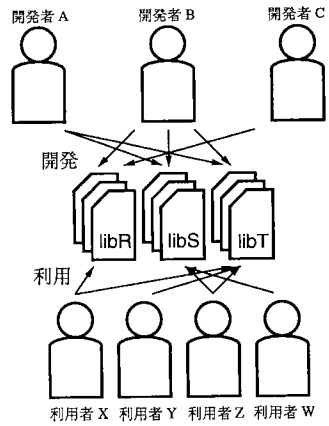

図 1　アプリケーション開発の例題
Fig. 1　An example of application development.

このような状況において，例えば以下のようなインタラクションが存在する．

- 共通ライブラリー利用者が新たなライブラリーを利用し始める

もしコンポーネントの開発者がこれまで利用してなかった共通ライブラリーを利用したいと思った際には，各種の告知を送ってもらうよう，該当する共通ライブラリーの開発者へ知らせておく．但し，自分の使っていない共通ライブラリーの告知を読む必要はないため，自分が使っている共通ライブラリーに関する告知だけを受け取る．

- バグを発見した場合にそれを報告する

共通ライブラリーの中にバグを発見した場合，それを該当するライブラリーの開発者へ報告する．また，報告されたバグがどう修正されたか（されなかったか）の結果が通知される．あるいは，だれかが既に発見したバグについても同様に通知される．

- 不明な点について問合せを行う．

共通ライブラリーの機能や利用方法についての質問を開発者に投げる．

この例はごく小さな開発チームが仮定されているにもかかわらず，多くの種類のインタラクションを挙げることができる．

本研究では，開発環境においてメッセージのやりとりやファイルの送受信等，計算機を用いて行われるインタラクションを支援するための，プロセス中心型インタラクションモデルを提案する．本モデルは，インタラクションのひな型のためにソフトウェアプロセス記述を用いている．モデルを表現するために，我々は作業エージェント（WorkAgent）と呼ぶインタラクションの主体となる物を導入する．作業エージェントはチャネル（Channel）を用いてインタラクションを行う．

2.3 関連研究

多くの既存のプロセス中心型開発環境は，上記で述べたようなインタラクションをパラメータの受渡しや単純なメッセージ送受信としてとらえている．このような環境では，一般的に実際にインタラクションを行う対象（例えば，だれがインタラクションの相手なのか）を具体的に把握する必要があるか，プロセス記述の抽象度が高いためのインタラクション自身を表現することが難しくなっている．

一方，既に提案，運用されているコミュニケーション支援ツールを導入してインタラクションの支援を行うとする場合，次のような問題が無視できない．

- インターネットで用いられている電子メールを用いるツールの場合，届けられるべき相手先に確実に届けられることを保証することが難しい．更に，利用者間の会話等に電子メールを用いるのはあまり適切ではない．
- talk や phone 等の対話ツールを用いることにより，利用者間の会話等には効果的な支援が行える．しかし，例えば一つのファイルのような比較的大きな情報を転送する場合にはあまり適さない．また，個々の対話が独立して行われるために，インタラクションの全体的な制御や記録を行うことは難しい．
- [5]〜[7],[14]等のグループウェアを分散データベースとして用いることによっても効果的な支援が行える．しかし，これらの研究では比較的固定した対話経路に対する支援を行うものが多く，ソフトウェア開発作業のように作業の結果や進ちょく状況によって動的に構成が変化する物には不適切であると考える．また，対象が人間同士のコミュニケーションを想定しているものが多い．

3. プロセス中心型インタラクションモデル

本章では，我々の提案するプロセス中心型インタラクションモデルについて説明する．まず最初に，モデルに対する要求事項をまとめた上でモデルの概要について説明する．次に，モデルで用いられる作業エージェントとチャネルの詳細について解説する．最後に，前述の例を用いて具体的なインタラクションの記述例や記述されたインタラクションの表現を示す．

3.1 必要とされる事項

提案するモデルは，ソフトウェアプロセスにおけるインタラクションを記述し支援するために次のような機能が必要となる．

- プロセス中心型であること

我々のモデルは，効率の良いインタラクションの方法を表現する必要がある．つまり，プロセスの構造は直接インタラクションの構造を直接表現できる必要がある．

- インタラクションの主体を考慮すること

開発環境中で行われるあらゆるインタラクションを把握するために，提案するモデルはインタラクションの主体，つまり，インタラクションを実際に行っている作業者等について考慮する必要がある．

- インタラクションの中身を考慮すること

モデルはインタラクションの中で扱われれている情報の内容もまた考慮しなければいけない．なぜならば，この種の情報は開発プロセスの管理等を効果的に行う上で非常に重要な対象となるからである．

- 簡潔な枠組みであること

我々はこのモデルが一般に用いられている，開発作業全体のすべての作業を支援することが目的である汎用のプロセスモデルにおいて一つのコンポーネントとなることを想定している．従って，インタラクションモデルは適度な粒度と簡潔さをもって，汎用のプロセスモデルにおける他のコンポーネントと統合できるものでなければならない．

3.2 モデル定義

3.2.1 概　　要

本モデルではまず，ソフトウェアプロセスをタスクの集合として定義する．各タスクはアクティビティの系列とする．アクティビティはソフトウェアプロセスにおける不可分の動作であり，各タスクは他のタスクと相互にメッセージを交している．

3.1 で述べた要求を満たすため，我々はオブジェクト指向風のモデルを採用した．モデルには作業エージェントとチャネルと呼ばれる二つのクラスがあり，これらを用いてモデルを構成する．

作業エージェントはインタラクションとなるイベントを発生される実体であり，そのインスタンスは，人間，プログラムあるいは外部のシステムによって実行されるタスクとなる．

チャネルはインタラクションの内容および種類に基づいてクラス分けされた仮想的な通信路である．そのインスタンスはソフトウェアプロセスにおけるある話題，例えば「単体テストとその実行結果」などを表現するために生成される．

作業エージェントは他の作業エージェントとチャネルを媒体としてインタラクションを行う．この際，作業エージェントは媒体として用いるチャネルに接続することによって情報の送受信を行う．

形式的には，我々のプロセス中心型モデルは以下のように定義できる．

```
インタラクション ::=
  [作業エージェントの集合, チャネルの集合, 接続]
作業エージェント ::= イベントを発生する有限状態機械
チャネル ::= 作業エージェントが用いる仮想通信路
接続 ::= チャネルから作業エージェントへの対応関数
```

以下では，作業エージェントとチャネルの詳細について述べる．

3.2.2 作業エージェント

インタラクションをモデル化するという観点で言えば，すべてのタスクはインタラクションを生成する実体であると考えられる．よって，タスクを作業エージェントとしてモデル化する．言い換えれば，作業エージェントは一連の実行の間メッセージのやりとりを行ってインタラクションを行っているものである．

作業エージェントは次のように定義される．

```
作業エージェント ::= [名前, 属性の集合, 作業内容の集合]
名前 ::= 文字列
属性 ::= [属性名, 属性値]
属性名 ::= 文字列
属性値 ::= スカラー値 (範囲はプロセス記述毎に決定)
作業内容 ::= [前条件, タスク]
前条件 ::= 入力に対して真偽値を返す関数
タスク ::= アクティビティの系列
アクティビティ ::= 開発中における不可分の動作
```

作業エージェントは名前，属性，作業内容から構成される．名前は作業エージェントを特定するために用いられる．属性は列挙型の変数であり，作業エージェントの行う作業の特性を表現するために用いられる．このため，各変数の値域についてはプロセスを記述するたびごとに定義される．作業内容はアクティビティの系列であり，その実行前には前条件を満たすことが要求される．プロセス中に存在するアクティビティについては各プロセスおよび作業エージェントごとに決定される．

3.2.3 チャネル

ソフトウェアプロセス中のインタラクションは，作業エージェントによって生成された一連のメッセージとして定義する．これらのインタラクションは，その内容，インタラクションを行う形態，関連する作業エージェント等に基づいて分類することができる．

従って作業エージェント間には，分類された作業エージェントによって交される一連のメッセージによってある関係が定義できる．本モデルではこの関係をチャネルとして定義する．

チャネルは次のように定義される．

```
チャネル ::= [名前, 属性の集合, 閉包, 情報形式の集合]
名前 ::= 文字列
属性 ::= [属性名, 属性値]
属性名 ::= 文字列
属性値 ::= スカラー値 (範囲はプロセス記述毎に決定)
閉包 ::= [内閉包, 外閉包]
内閉包 ::= 作業エージェントで定義される属性値の組
(外閉包は内閉包と同様に定義される)
情報形式 ::= [型名, フィールド宣言の集合]
型名 ::= 文字列
フィールド宣言 ::= [フィールド名, フィールド型]
フィールド名 ::= 文字列
フィールド型 ::= 文字列, 論理値など
```

チャネルは名前，属性，閉包，情報形式から構成される．名前と属性については作業エージェントと同様に定義され利用される．

閉包はインタラクションを行う範囲を作業エージェントの型を用いて制御するために定義される．閉包には内閉包と外閉包の2種類がある．内閉包はチャネルに必ず接続していなければならない作業エージェントを指定し，外閉包はチャネルに接続できない作業エージェントを指定する．閉包の指定は作業エージェントのもつ属性の組で行う．これにより，例えば内閉包を用いることによって必要なインタラクションを定義でき，外閉包を用いることによって情報を隠ぺいしたい範囲を定義することができる．情報形式はチャネルで交されるメッセージの型定義であり，型の名前とメッセージの構成を宣言するフィールド宣言から構成される．

作業エージェントはチャネルへの接続/切断動作をチャネルを用いてインタラクションを行う前/インタラクションが終了した後に必ず行う．これによって，チャネルへの接続状態が現在の作業エージェントが関連するインタラクションの数および種類を表現できる．

3.3 記述の具体例

ここでは，具体的にモデルによるインタラクションの表現方法について紹介し，どのように作業エージェントやチャネルが動作するかを説明する．このために，2.2で用いたアプリケーション開発の例題を再度用い

ることとする．

まず図 2 は，この例題をモデルを用いて表現したものである．

この図では，角の取れた四角形が作業エージェントを示し，二重円がチャネルを示す．この図では，七つの作業エージェントと三つのチャネルが書かれている．作業エージェントとチャネル間に引かれた点線は，作業エージェントとチャネルとの接続を示す．以下では，これらの記述を行う手順の詳細を順を追って説明する．

3.3.1 作業エージェントおよびチャネルの特定

例題のような複数のライブラリーを用いた開発の際に発生するインタラクションには，さまざまなものが考えられる．例えば，特定のライブラリーに関して「開発者間の連絡」「開発者と利用者間における情報交換」「開発中ライブラリーの流通」といったインタラクションが考えられる．これらを表現するためには，これらの分類したい項目それぞれにチャネルを定義することになる．また，「ライブラリー中で定義される関数の利用方法」のようなインタラクションは，各ライブラリーの相互作用等が話題になることも考えられるため，ライブラリーの数に依存せず全体で一つチャネルを定義すれば十分であろう．このように，本モデルでは，ある状況に対して複数のモデルによる記述を行うことが可能である．ここでは，記述をわかりするために，単一のプロダクトに対して一つのチャネルを割り当て，それぞれのライブラリーに関するインタラクションを単一のチャネルで表現することとする．

この例題では，共通ライブラリーの開発者と利用者との間でインタラクションが発生すると考えられる．よって，ここでは開発者および利用者をそれぞれ作業エージェントに対応づけることとする．インタラクションの内容としては互いに開発/利用している共通ライブラリーに関する事項となるであろうことから，ここではライブラリーの名前をチャネルに対応づけた(図 2)．

各作業エージェントからチャネルへの接続は，それぞれの開発者/利用者が関連している共通ライブラリーへの関係を示すように引かれている．つまり，開発者を示す作業エージェントはその開発者が担当している共通ライブラリーのチャネルへ，また，利用者を示す作業エージェントはその作業者がコンポーネントの開発の際に利用している共通ライブラリーのチャネルへ引かれている．

3.3.2 作業エージェントの記述

図 3 は開発者 A に対応する作業エージェントの記述例である．

作業者 A の属性として，ここでは役割と主担当という二つの属性を定義した．まず役割は「開発プロセスにおけるこの作業エージェントの役割分担」を示し，「開発者」「利用者」の二つの値をとるものとした．主担当は「作業エージェントが主にかかわるライブラリー」を意味し，開発者を表す作業エージェントであれば自らが開発責任者となっているライブラリーの名前，利用者を表す作業エージェントでは自らが利用するライブラリーのうち一番利用頻度の高いライブラリーの名前を表現するものとした．

作業エージェントが実際に動作するための記述を行う

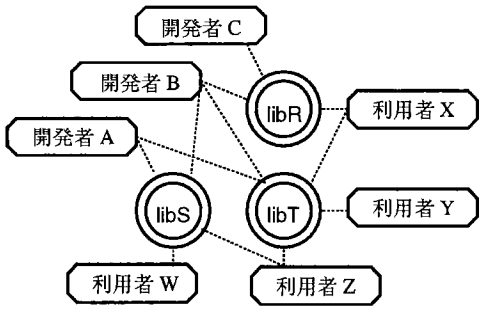

図 2 モデルを用いた例題の表現
Fig. 2 An expression of example with the model.

```
define workagent Developer_A
begin
  attribute begin
    role: develop;
    primary: S;
  end
  vars begin
    bugbuffer: array of bug-report;
    qbuffer: array of question; res: bool;
  end
  initaction begin join(libS); join(libT); end
  primitive add consult, debugging, makeans;
  action begin
    channel libS begin
      bug-report:
        push(bugbuffer, info); res=consult(info);
        if (res == True) { send(libS, debugging(info); }
      question:
        push(qbuffer, info); send(libS, makeans(info));
    end
    channel libT begin
      bug-report:
        res = consult(info);
        if (res == True) { send(libT, debugging(info); }
      question:
        send(libT, makeans(info));
    end
  end
end
```

図 3 作業エージェントの記述
Fig. 3 A description of WorkAgent.

ために，3.2.2で述べた内容（attribute，action節）の他に，変数定義（vars節），初期化動作（initaction節），基本動作宣言（primitive節）等の記述を行う．

変数定義は作業エージェント内部で情報やインタラクションを保持するために用いられる変数を定義するものである．初期化動作では作業エージェントが動作を開始する際に行っておく動作を定義する．この例では，2種類のチャネルへの接続動作が定義されている．基本動作の宣言ではどのような動作が記述する作業エージェントにおけるアクティビティであるかを宣言するが，その動作の具体的内容はモデルの考慮外であるために記述されない．これにより，同じ基本動作宣言を記述しながら，作業エージェントが動作する環境によって実際の挙動を変更することが可能となるため，作業エージェント記述の可搬性を高くすることができる．

3.3.3 チャネルの記述

図4は，共通ライブラリー *libS* に対するチャネルの記述例である．

作業エージェントと同様，チャネルの属性も定義される．ここでは，ライブラリー名が属性として定義されている．3.2.3で述べた属性，閉包，情報形式はそれぞれattribute，restriction，infotype節として記述されている．

この例では，チャネルに対して内閉包の定義があり「共通ライブラリー *libS* の開発者は必ずこのチャネルに接続していなければならない」ことを記述している(注1)．また，四つの型名が定義されており，それぞれ「告知」「バグ報告」「質問」「回答」を示している．各型名には複数のフィールド宣言があり，話題を示す文字列やキーワード，メッセージの書かれたファイルなどの存在が定義されている．

```
define channel libS
begin
   attribute begin
      libname: S;
   end
   restriction
      in: (develop, S)
   end
   infotype begin
      announce:
         (topic: string, contents: file);
      bug-report:
         (topic: string, contents: file);
      question:
         (topic: string, body: file, keyword: string);
      answer:
         (topic: string, body: string);
   end
end
```

図4　チャネルの記述
Fig. 4　A description of channel.

3.3.4 作業エージェントとチャネルの振舞い

図2で示されている作業エージェントとチャネル定義や接続状況は定義できているため，これを用いて作業エージェントやチャネルの記述が実際にどのように動作するか，いくつかのシナリオを用いて説明する．

（a）シナリオ1：新しいライブラリーの利用

利用者 Y が新たに共通ライブラリー *libR* を利用し始める状況を考える．この場合，利用者 Y に対応する作業エージェントで“join (*libR*)”という動作を行う．これによって，この作業エージェントが *libR* を利用することを示す．また，これによって *libT* の利用を中止する場合には，“leave (*libT*)”という動作を行ってチャネル *libT* からの切断を行うことになる．

（b）シナリオ2：バグの発見

開発者 B が共通ライブラリー *libS* にバグを発見した状況を考える．まず，開発者 B はバグ報告をチャネル *libS* に送信する．このバグ報告はチャネル *libS* に接続しているすべての作業エージェント，つまり，利用者 W，Z と開発者 A に送信される．開発者 A は共通ライブラリー *libS* の開発責任者であるため，A は送られたバグ報告をバッファに蓄え，これが本当にバグであるかの検証を行う．その結果バグであることが判明したため，開発者 A はデバッグを行ってその結果をチャネル *libS* に送信する．

なお，ライブラリーの開発者 B だけでなく，チャネル *libS* に接続しているすべての作業エージェントがチャネルを用いてバグ報告を送信することができ，その報告もまた同様にチャネル *libS* に接続しているすべての作業エージェントが受信することができる．これにより，共通ライブラリーの利用者は情報を迅速に把握することができる上，送信する側は送信相手を意識することなく適切なインタラクションを行うことができる．

（c）シナリオ3：問合せの送信

シナリオ2と同様，ある共通ライブラリーに対する質問はすべての開発者に対して送信され，その回答はその共通ライブラリーの利用者だけに確実に届けられる．これらの質問/回答はチャネルに接続された作業エージェント間で共有されており，従って重複した質問などの無駄なインタラクションを排除できる．

(注1)：外閉包を定義する場合，例えば「開発者は接続してはいけない」チャネルを定義する際には，対象となる開発者を示す作業エージェントがもつ属性の組を指定する．

4. モデルに基づいた開発環境

本章では，3. で提案したモデルに基づいたソフトウェア開発支援環境について述べる．まず支援環境の概要について述べ，次にその実装方法を示す．

4.1 概　　要

モデルに基づいた支援環境は以下のような機能を提供する．

- 作業エージェント間のインタラクション実行

ネットワーク環境下で，記述された作業エージェントやチャネルが実際に機能することができる．

- 作業エージェントの実行支援

作業エージェントで行われる作業が計算機によって実行可能であった場合に，それを自動的に実行する．

- ソフトウェア開発者に対する作業誘導

作業エージェントで行われる作業が人手によるものであった場合，作業に必要なツールの実行などを行う．また，作業エージェントのもつ情報を統合して現在の作業スケジュールを示し，次に行える作業の誘導を行う．

4.2 構　　成

図 5 は今回設計した支援環境の構成を表したものである．

本支援環境は比較的小規模のネットワーク上で用いらえることを想定しており，次のような部分から構成される．

- 作業エージェント代理プログラム
- チャネルサーバ
- 作業エージェントマネジャ

各作業エージェントはその代理プログラムによって構築される．代理プログラムは作業エージェントの定義を読み記述された動作を代行する．すべてのチャネルはチャネルサーバによって制御される．代理プログラムとチャネルサーバはネットワーク越しに TCP/IP 接続されている．

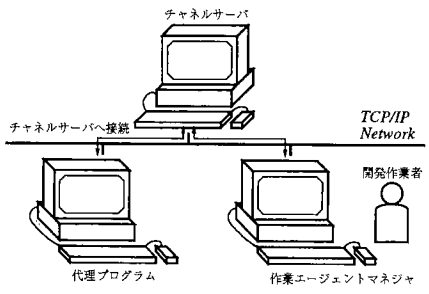

図 5　支援環境の構成
Fig. 5　A structure of supporting environment.

作業エージェントマネジャは複数の代理プログラムを統轄し，ソフトウェア開発者に対するユーザインタフェースとなる．作業エージェントマネジャは現在の代理プログラムの状況を示し，ツール起動などの作業誘導を行う．

作業エージェントマネジャに管理されない代理プログラムは単体で動作を行う．生成物管理ツールやプロジェクト管理ツール等のような外部プログラムの実行，事前に定義された作業の実行などだけを行う作業エージェントはこのような形で実現される．

チャネルサーバは代理プログラムとの接続を管理し，受けとった情報を適切な代理プログラムへ送出する．チャネルサーバを通過するインタラクションの内容は履歴として保存する．保存された履歴を用いて，開発作業の進ちょく状況の分析等を行うことができる．

4.3 システムの試作

我々は本システムの試作を行った．試作したシステムは BSD UNIX 上で C 言語を用いて実装された．代理プログラムは作業エージェントの定義を読み込み，チャネルサーバと TCP/IP で接続し，メッセージの送受信を行うことができる．

試作した代理プログラムは単一のファイルや文字列といった単純なメッセージの送受信を行う．作業エージェントマネジャの実装を行っていないため，代理プログラムが自分自身で端末型ユーザインタフェースをもち，ソフトウェア開発作業者からの操作を受け付ける．代理プログラムがチャネルからファイルを受信した場合にはそれをあらかじめ定められた場所に保存し，ファイルの到着を開発作業者へ知らせる．文字列を受信した場合にはそれをそのまま画面へ出力する．

試作したチャネルサーバは，ソケット処理エンジン，チャネル管理テーブル，ソケット管理テーブルなどから構成されており，代理サーバからの接続を受け付け，メッセージの送受信を管理，記録する．

試作した代理プログラムとチャネルサーバを 3. で記述したアプリケーション開発の例題に適用しその動作を確認した．

5. む　す　び

本研究では，ソフトウェア開発環境におけるインタラクションを表現するプロセス中心型インタラクショ

ンモデルの提案を行った．本モデルは作業エージェントとチャネルと呼ばれる二つの要素から構成される．これにより，インタラクションはチャネルを媒体して作業エージェントによって生成された一連のイベントとして定義される．本モデルを用いることにより，開発環境中のインタラクションが明確に特定され記述できることがわかった．

また，本研究ではモデルに基づいた開発支援環境の設計を行い，その試作を行った．本支援環境はモデルにて定義されたインタラクションを実際に動作させることができ，自動実行支援，ツール起動や作業誘導を行うことができる．試作したシステムを用いて，文字列やファイルの送受信を行って実際に開発作業者間でのインタラクションが行えることを確認した．

今後の課題として，更に実験や実装を重ねることによってモデルおよび支援環境の評価や，更に高度なインタラクションモデルによるより進んだ作業者間のコミュニケーション支援を行う枠組みについて考察を行うことが挙げられる．また，支援環境の実装にあたっては，記述された内容の依存関係を追跡することによって，記述の一貫性を確認しそれをわかりやすく表示することが必要であると考えられる．

文　献

[1] 青山幹雄，“分散開発環境：新しい開発環境像をもとめて，”情報処理，vol.33, no.1, pp.2–13, 1992.

[2] 井上克郎，“ソフトウェアプロセスの研究動向，”ソフトウェア科学会ソフトウェアプロセス研究会報告，95–SP–2–1, pp.1–10, 1995.

[3] G. Kaiser, P. Feiler, and S. Popovich, “Intelligent assistance for software development maintenance,” IEEE Software, vol.5, no.3, pp.40–49, 1988.

[4] T. Katayama, “A hierarchical and functional software process description and its enaction,” Proc. of 11th Int. Conf. on Software Eng., pp.343–352, 1989.

[5] K. Lai and T.W. Malone, “Object lens: A “spreadsheet” for cooperative work,” Proc. of CSCW ’88, pp.115–124, 1988.

[6] T.W. Malone, K.R. Grant, K-Y. Lai, R. Rao, and D. Rosenblitt, “Semistructured messages are surprisingly useful for computer-supported coordination,” ACM Transactions on Office Information Systems, vol.5, no.2, pp.115–131, 1987.

[7] T.W. Malone, K. Lai, and C. Fry, “Experiments with oval: A radically tailorable tool for cooperative work,” Proc. of CSCW ’92, pp.289–297, 1992.

[8] 松下　誠，飯田　元，井上克郎，鳥居宏次，“開発プロセスを構成する要素間のインタラクションに関する考察，”情処研報，95–SE–102, pp.165–170, 1995.

[9] M. Matsushita, H. Iida, and K. Inoue, “An interaction support mechanism in software development,” Proc. of 1996 Asia-Pacific Software Engineering Conference, pp.66–73, 1996.

[10] 落水浩一郎，“ソフトウェアプロセスに関する研究の現状，”ソフトウェア科学会ソフトウェアプロセス研究会，SP93–1–1, pp.1–11, 1993.

[11] L. Osterweil, “Software processes are software too,” Proc. of 9th Int. Conf. Software Eng., pp.2–13, 1987.

[12] S. Sutton, D. Heimbigner, and L. Osterweil, “Language constructs for managing change in proces-centered environments,” Proc. of 9th Int. Conf. Software Eng., pp.328–338, 1987.

[13] ソフトウェアプロセスシンポジウム論文集，情報処理学会，1994.

[14] 特集“グループウェアの実現に向けて，”情報処理，vol.34, no.8, 1993.

付　録

1. 作業エージェントおよびチャネルの記述言語

本付録では，今回試作したシステムにおいて，作業エージェントおよびチャネルを記述する際に用いられる言語の言語仕様をEBNFにて示す．また，各言語要素の意味について簡単に説明する．なお，記述全体は以下に述べる言語仕様における非終端記号InteractionDescrsにて表される．

1.1 言語仕様

```
InteractionDescrs ::- {InteractionDescr}.
InteractionDescr ::- "define" WorkAgentOrChannel.
WorkAgentOrChannel ::- WorkAgentDescr | ChannelDescr.

WorkAgentDescr ::-
        "workagent" WorkAgentName WorkAgentDescrBody.
ChannelDescr ::-"channel" ChannelName ChannelDescrBody.
WorkAgentName ::- NAME.
ChannelName ::- NAME.
WorkAgentBody ::- "begin" WorkAgentElements "end".
ChannelBody ::- "begin" ChannelElements "end".
WorkAgentElements ::-
        AttributeElement [VarsElement]
        [InitActionElement] [PrimitiveElement]
        ActionElement.
ChannelElements ::- AttributeElement RestrictionElement
                    InfoTypeElement.

AttributeElement ::- "attribute" AttributeContents.
AttributeContents ::-
        "begin" AttributeContent
        {AttributeContent} "end" | AttributeContent.
AttributeContent::-AttributeName ":" AttributeValue ";".
AttributeName ::- NAME.
AttributeValue ::- NAME.

VarsElement ::- "vars" VarsContents.
VarsContents::-"begin" VarsContent {VarsContent} "end"|
                VarsContent.
VarsContent ::- VarsName ":" VarsType ";".
```

```
VarsName ::- NAME.
VarsType ::- BuiltinType | InfoTypeType | ArrayType.
BuiltinType ::- "integer" | "number" | "string" |
               "file" | "boolean".
ArrayType ::- "array of" ArrayComposeType.
ArrayComposeType ::- BuiltinType | InfoTypeType.

InitActionElement ::- "initaction" InitActionContents.
InitActionContents ::- "begin" InitActionContent
                       {InitActionContent} "end" |
                       InitActionContent.
InitActionContent ::- ActionBody {ActionBody}.

PrimitiveElement ::- "primitive" PrimitiveContents.
PrimitiveContents ::- "begin" PrimitiveContent
                      {PrimitiveContent} "end" |
                      PrimitiveContent.
PrimitiveContent ::- "add" PrimitiveAddSymbol
                     {"," PrimitiveAddSymbol}.
PrimitiveAddSymbol ::- NAME.

ActionElement::-"action" ActionContent {ActionContent}.
ActionContent ::- "channel" ActionChannelName "begin"
                  ActionWithTag "end".
ActionChannelName ::- NAME.
ActionWithTag ::- ActionTag ":" ActionBody.
ActionTag ::- InfoTypeType.
ActionBody::-ActionStatement {";" ActionStatement} ";".
ActionStatements ::- "begin" ActionBody "end"
ActionStatement ::- ActionBasic | ActionConditional
ActionConditional ::-
        "if" "(" Expression ")" "{" ActionTBody "}"
        "else" "{" ActionBody "}" |
        "if" "(" Expression ")" "{" ActionBody "}"
        "while" "(" Expression ")" "{" ActionBody "}"
ActionTBody ::- ActionTStatement
               {";" ActionTStatement} ";".
ActionTStatement ::- ActionBasic | ActionTConditional
ActionTConditional ::-
        "if" "(" Expression ")" "{" ActionTBody "}"
        "else" "{" ActionTBody "}" |
        "while" "(" Expression ")" "{" ActionTBody "}"
ActionBasic ::- ActionEquation | ActionFuncall |
               ActionStatements | ActionNull.
ActionNull ::-.
ActionEquation ::- ActionEquLeft "=" Expression.
ActionEquLeft ::- VarsName.
ActionFuncall ::- FuncName "(" [Expressions] ")".
FuncName ::- NAME.
Expressions ::- Expression {"," Expression}.
Expression ::- SimpleExpr [RelationOpr SimpleExpr].
SimpleExpr ::- [Flag] Term {AddOpr Term}.
Term ::- Factor {MultOpr Factor}.
Factor ::- VarsName | Constant |
          "(" Expression ")" | ActionFuncall.
RelationOpr ::- "==" | "!=" | ">" | ">=" | "<=" | "<".
AddOpr ::- "+" | "-" | "||".
MultOpr ::- "*" | "/" | "&&".
Flag ::- "+" | "-".
Constant ::- SimpleNumber | String | "True" | "False".
SimpleNumber ::- NUM {NUM}.
String ::- """ StringContent """.
StringContent ::-
        arbitally character except double-quotation.

RestrictionElement::-"restriction" RestrictionContents.
RestrictionContents ::-
        [InclosureContents] [ExclosureContents].
InclosureContents ::- "in:" InclosureElement
                      {"," InclosureElement}.
InclosureElement ::- "(" AttributeSymbols ")".
ExclosureContents ::- "out:" ExclosureElement
                      {"," ExclosureElement}.
ExclosureElement ::- "(" AttributeSymbols ")".
AttributeSymbols ::-
        AttributeSymbol {"," AttributeSymbol}.
AttributeSymbol ::- NAME.

InfoTypeElement ::- "infotype" InfoTypeContents.
InfoTypeContents ::- "begin" InfoTypeContent
                     {InfoTypeContent} "end" |
                     InfoTypeContent.
InfoTypeContent ::-
        InfoTypeType ":" "(" FieldSymbols ")" ";".
InfoTypeType ::- NAME.
FieldSymbols ::- FieldSymbol {"," FieldSymbol}.
FieldSymbol ::- FieldName ":" FieldType.
FieldName ::- NAME.
FieldType ::- VarsType.

NAME ::- ALPH { ALPH | NUM }.
ALPH ::- "a" | ... | "z" | "A" | ... | "Z" | "_".
NUM ::- "0" | ... "9".
```

1.2 言語要素の意味

1.2.1 作業エージェント（WorkAgentDescr）

- 属性（AttributeElement）

実行時には，記述されるすべての作業エージェントにおいて，同じ属性名が定義されているか，定義された順番が同一であるかどうかが確認される．この定義された順番は，チャネルの記述における閉包の定義を記述する際に，属性値の組に書かれる順番と対応づけられる．

- 変数定義（VarsElement）

作業内容の記述の際に，一時的な情報を蓄えておくための変数は宣言されなければならない．変数は定義された作業エージェント内でのみ有効となる．変数には型があり，組込みの型（整数，実数，文字列，ファイル，論理値）か，若しくはチャネルにおける情報形式の型を用いることができる．変数はその初期値として数字ならば 0，文字列やファイルならば空文字列，論理値ならば偽をとる．

- 初期化動作（InitActionElement）

初期化動作は作業エージェントの動作開始時に一度だけ評価される．この時点では，作業エージェントはどのチャネルにも所属していないため，他の作業エー

ジェントからの情報を受け取ることはできないが，明示的にチャネルに所属することによって，チャネルへ情報を送ることは可能である．初期化動作として記述された動作を終了する際には，各チャネルに記述された内閉包の条件が調べられる．もし，条件に該当するチャネルに接続していなかった場合には，初期化動作として該当するチャネルへの接続動作が自動的に行われる．

- 基本動作宣言（PrimitiveElement）

宣言される動作は，実行系においてあらかじめその動作内容が定義されていなければならない．基本動作宣言に記述されているが，その内容が定義されていない動作を実行した場合には，実行時にエラーとなり処理は行われない．

- 作業内容（ActionElement）

前条件として，現時点ではチャネル名と情報形式の組で指定することによって特定のチャネルに特定の情報が流れた場合を指定することができる．作業内容の記述中では，変数名 info にチャネルより流れてきた情報の内容が収められている．作業内容自体の記述は一般に見られる簡単な手続き型言語と同様に，数値演算や制御構造を記述することができる．アクティビティは関数呼出しの形で記述される．アクティビティとしては，組込みのアクティビティである「チャネルへの接続（join）」「チャネル接続の切断（leave）」「チャネルへの情報送出（send）」のほか，基本動作宣言にて宣言された動作がある．send を用いる際には，引数で明示しない限り，その記述の前条件で指定されるチャネルあるいは情報形式の情報を送出すると解釈される．

1.2.2 チャネル（ChannelDescr）

- 属性（AttributeElement）

作業エージェントのときと同様に，同じ属性名が定義されているかが動作時に確認される．

- 閉包（RestrictionElement）

属性値の組として並べた順番と，作業エージェントにおける属性の定義の順番は一致しているとして，閉包の指定は解釈される．つまり，作業エージェントの記述の際，A，Bという属性をこの順に定義した場合には，閉包の定義は二つの属性値の組（x，y）として記述され，それぞれ属性Aとしてx，属性Bとしてyを指定したとみなす．作業エージェントがチャネルへ明示的に接続する際，あるいは接続を切断する際には，該当チャネルの外閉包，内閉包として指定された値を調べる．これにより，作業エージェントがチャネルへの接続が許されるか，あるいはチャネルからの切断が許されるかの判断が実行時に行われる．

- 情報形式（InfoTypeElement）

フィールドの型は，作業エージェントの変数定義における組込みの型と同一の物が用いられる．チャネルが動作する際には，作業エージェントから送られてくる情報が事前に定義されている情報形式と一致しているか確認される．もし一致しない場合には，その情報は破棄される．

（平成9年10月8日受付，10年2月9日再受付）

松下　誠

平5阪大・基礎工・情報卒．平7同大大学院博士前期課程了．平10阪大・基礎工・情報数理・助手．ソフトウェア開発プロセスの研究に従事．

飯田　元　（正員）

昭63阪大・基礎工・情報卒．平2同大大学院博士前期課程了．同年同後期課程入学．平3阪大・基礎工・情報・助手．平7奈良先端大・情報科学センター・助教授．工博．ソフトウェア開発プロセスおよび開発支援環境，協調作業支援技術などの研究に従事．

井上　克郎　（正員）

昭54阪大・基礎工・情報卒．昭59同大大学院博士課程了．同年同大・基礎工・情報・助手．昭59～61ハワイ大マノア校・情報工学科・助教授．平1阪大・基礎工・情報・講師．平3同学科・助教授．平7同学科・教授．工博．ソフトウェア工学の研究に従事．