

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 HIDA^{††}, 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なう。

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なう。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なった。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成り、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なう。

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. はじめに

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、従来の一箇所に集中して行っていた開発とは異なり、進捗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なわれている [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はかっている。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] などがあげら

れる。

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け渡しといったインタラクションに着目して捉えた場合、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な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、もしくは単純な作業の型の1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充分ではない。

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なう [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な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わす。

以下 2. 節では、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. 節では、インタラクションのモデル化方法について説明する。4. 節で

[†] 大阪大学基礎工学研究科, 大阪府

Graduate School of Engineering Science, Osaka University,
1-3 Machikaneyama-cho, Toyonaka, Osaka 560 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, Nara 630-01
JAPAN

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. で本研究について纏める．

2. ソフトウェアプロセスとインタラクション

本節では、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なえる．

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捗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な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「生成物の受け渡し」「作業の開始/終了通知」「共同作業間での会話」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渡る．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なわれているとする．このアプリケーションは共通ライブラリ（*libR*、*libS*、*libT*の3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．

共通ライブラリを開発を行なう作業者が3名（*A*、*B*、*C*）おり、各ライブラリはそれぞれ複数人で開発を行なっており、開発責任者があらかじめ決定されているとする．ライブラリを開発責任者はライブラリを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、共通ライブラリを利用する作業者が4名（*X*、*Y*、*Z*、*W*）おり、各開発者は（1つ以上の）共通ライブラリを利用してコンポーネントの開

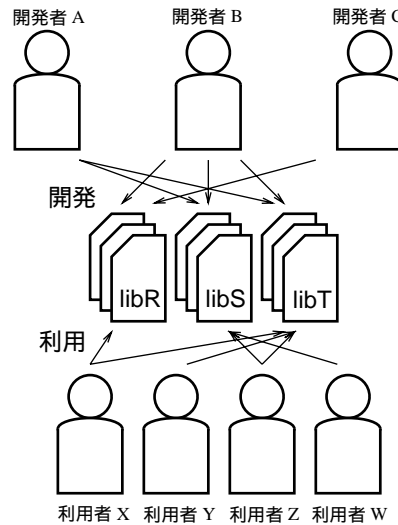


図1 アプリケーション開発の例題

Fig. 1 An example of application development

発を行なっている．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を開発者に対して質問を行ったり、他のライブラリの利用者とライブラリに関する情報交換を行ないたいとも思っている．

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．

- 共通ライブラリ利用者が新たなライブラリを利用し始める

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を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に関する告知だけを受けとる．

- バグを発見した場合にそれを報告する
- 共通ライブラリの中にバグを発見した場合、それを該当するライブラリを開発者へ報告する．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、誰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い合わせを行なう．

共通ライブラリ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関わらず、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、計算機を用いて行な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ンネル (Channel) を用いてインタラクションを行なう。

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け渡しや単純なメッセージ送受信として捉えている。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なう対象 (例えば、誰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なうとする場合、次のような問題が無視できない。

- インターネットを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。さらに、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なえる。しかし、例えば1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なわれるために、インタラクションの全体的な制御や記録を行なうことは難しい。

- [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効果的な支援が行なえる。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なうものが多く、ソフトウェア開発作業のように作業の結果や進捗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られる。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

本節では、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纏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ン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

開発環境中に行な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な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なう上で非常に重要な対象となるからである。

- 簡潔な枠組であること

我々はこのモデルが一般に用いられている、開発作業全体の全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1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持って、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 節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ンネルと呼ばれる2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。

チャン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ンネルを媒体としてインタラクションを行なう。この際、作業エージェントは媒体として用いるチャンネルに接続することによって情報の送受信を行なう。

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, チャンネルの集合, 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
チャンネル ::= 作業エージェントが用いる仮想通信路
接続 ::= チャン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ン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、全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なってインタラクションを行な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な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 チャンネル

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なう形態、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ンネルとして定義する。

チャンネルは次のように定義される。

```

チャンネル ::= [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 ::= 文字列
フィールド型 ::= 文字列, 論理値など
  
```

チャンネルは名前、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な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。内閉包はチャン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、外閉包はチャン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持つ属性の組で行なう。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蔽したい範囲を定義することができる。情報形式はチャン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ンネルへの接続/切断動作をチャンネルを用いてインタラクションを行なう前/インタラクションが終了した後には必ず行なう。これによって、チャン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。

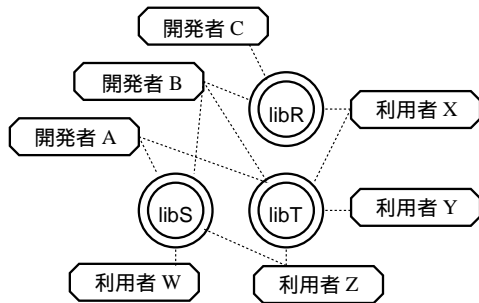


図2 モデルを用いた例題の表現

Fig. 2 An expression of example with the model

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ンネルが動作するかを説明する。このために、2.2節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、二重円がチャンネルを示す。この図では、7つの作業エージェントと3つのチャンネルが書かれている。作業エージェントとチャンネル間に引かれた点線は、作業エージェントとチャンネルとの接続を示す。以下では、これらの記述を行なう手順の詳細を順を追って説明する。

3.3.1 作業エージェント及びチャンネルの特定

例題のような複数のライブラリ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」「開発中ライブラリの流通」といったインタラクションが考えられる。これら表現するためには、これらの分類したい項目それぞれにチャンネルを定義することになる。また、「ライブラリ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、ライブラリの数に依存せず全体で1つチャン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な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1つのチャンネルを割り当て、それ

```

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
end

```

図3 作業エージェントの記述

Fig. 3 A description of WorkAgent

ぞれのライブラリに関するインタラクションを単一のチャン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付けることとする。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の名前をチャンネルに対応付けた(図2)。

各作業エージェントからチャンネルへの接続は、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のチャンネルへ、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のチャン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2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2つの値を取るものとした。主担当は「作業エージェントが主に関わるライブラリ」

```

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
end

```

図4 チャンネルの記述
Fig. 4 A description of Channel

を意味し、開発者を表わ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の名前、利用者を表わす作業エージェントでは自らが利用するライブラリのうち一番利用頻度の高いライブラリ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なうために、3.2.2節で述べた内容 (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) 等の記述を行なう。

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なっておく動作を定義する。この例では、2種類のチャン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 チャンネルの記述

図4は、共通ライブラリ *libS* に対するチャンネルの記述例である。

作業エージェントと同様、チャンネルの属性も定義される。ここでは、ライブラリ名が属性として定義されている。3.2.3節で述べた属性、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、チャンネルに対して内閉包の定義があり「共通ライブラリ *libS* の開発者は必ずこのチャン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、4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。

3.3.4 作業エージェントとチャンネルの振舞い

図2で示されている作業エージェントとチャン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ン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: 新しいライブラリの利用

利用者 *Y* が新たに共通ライブラリ *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*)” という動作を行なう。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*)” という動作を行なってチャンネル *libT* からの切断を行なうことになる。

b) シナリオ2: バグの発見

開発者 *B* が共通ライブラリ *libS* にバグを発見した状況を考える。まず、開発者 *B* はバグ報告をチャンネル *libS* に送信する。このバグ報告はチャンネル *libS* に接続している全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 *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なう。その結果バグであることが判明したため、開発者 *A* はデバッグを行なってその結果をチャンネル *libS* に送信する。

なお、ライブラリの開発者 *B* だけでなく、チャンネル *libS* に接続している全ての作業エージェントがチャン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ンネル *libS* に接続している全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なうことができる。

(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」チャン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持つ属性の組を指定する。

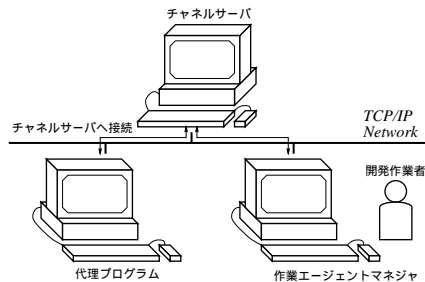


図5 支援環境の構成

Fig. 5 An structure of supporting environment

c) シナリオ3: 問いあわせの送信

シナリオ2と同様、ある共通ライブラリに対する質問は全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/回答はチャン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。

4. モデルに基づいた開発環境

本節では、3. 節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ン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な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な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なう。また、作業エージェントの持つ情報を統合して現在の作業スケジュールを示し、次に行なえる作業の誘導を行なう。

4.2 構成

図5は今回設計した支援環境の構成を表わしたものである。

本支援環境は比較的小規模のネットワーク上で用いられ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
- チャンネルサーバ
- 作業エージェントマネージャ

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。全てのチャンネルはチャンネルサーバによって制御される。代理プログラムとチャンネルサーバはネットワーク越しにTCP/IP接続されている。

作業エージェントマネー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ーフェースとなる。作業エージェントマネー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なう。

作業エージェントマネージャに管理されない代理プログラムは単体で動作を行なう。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なう作業エージェントはこのような形で実現される。

チャン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。チャン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、開発作業の進捗状況の分析等を行なうことができる。

4.3 システムの試作

我々は本システムの試作を行なった。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こみ、チャンネルサーバと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なう。作業エージェントマネー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ーフェースを持ち、ソフトウェア開発業者からの操作を受けつける。代理プログラムがチャン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。

試作したチャンネルサーバは、ソケット処理エンジン、チャンネル管理テーブル、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つけ

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ンネルサーバを 3. 節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. ま と め

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なった．本モデルは作業エージェントとチャンネルと呼ばれる 2 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ン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ない，その試作を行なった．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なうことができる．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なって実際に開発作業者間でのインタラクションが行なえることを確認した．

今後の課題として，さら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，さら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なう枠組について考察を行な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”, In Proceedings 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”, Proceedings of CSCW'88, pp.115-124, 1988.
- [6] T.W. Malone, et al., “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”, Proceedings 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”，情報研報，95-SE-102，pp.165-170，1995．
- [9] M. Matsushita, H. Iida, and K. Inoue, “An Interaction Support Mechanism in Software Development”, In Proceedings 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”, In Proceedings of 9th Int. Conf. Software Eng., pp.2-13, 1987.
- [12] S. Sutton, D. Heimbigner, and L. Osterweil, “Language Constructs for Managing Change in Process-Centered Environments”, In Proceedings 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. 作業エージェント及びチャンネルの記述言語

本付録では，今回試作したシステムにおいて，作業エージェント及びチャン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 にて表わされる．

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 :-
    arbitrarily 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、記述される全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を確認される。この定義された順番は、チャン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付けられる。

- 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、組み込みの型 (整数, 実数, 文字列, ファイル, 論理値) か、もしくはチャン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, 論理値ならば偽を取る。

- 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ン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とることはできないが、明示的にチャンネルに所属することによって、チャン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、各チャン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ン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ンネルへの接続動作が自動的に行なわれる。

- 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いと見なされる。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なわれない。

- 作業内容 (ActionElement)

前条件として、現時点ではチャン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ン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 にチャンネルより流れてきた情報の内容が納められている。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び出しの形で記述される。アクティビティとしては、組み込みのアクティビティである「チャンネルへの接続 (join)」「チャンネル接続の切断 (leave)」「チャンネルへの情報送出 (send)」の他、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ン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 チャンネル (ChannelDescr)

- 属性 (AttributeElement)

作業エージェントの時と同様に、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 2 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ン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、該当チャンネルの外閉包、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ンネルへの接続が許されるか、あるいはチャンネルからの切断が許されるかの判断が実行時に行なわれる。

- 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み込みの型と同一の物が用いられる。チャン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 同大学大学院博士前期課程修了。平 10 年阪大・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 同大学大学院博士前期課程修了。同年同後期課程入学。平 3 年阪大・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 同大学大学院博士課程修了。同年同大・基礎工・情報・助手。昭 59 ~ 昭 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。