ソフトウェア工学 82－1
（1991. 12. 3）

ソフトウェアの分散開発のモデル化の試み

岡田世志彦[1] 飯田元[1] 井上克郎[1] 鳥居宏次[1] 永岡渡[2] 梅本肇[2] 酒井睦雄[3]

[1]大阪大学基礎工学部 [2]日立西部ソフトウェア（株） [3]（株）日立製作所

複数の作業チームが離れた場所で連絡を取り合いながら開発を行う，ソフトウェア分散開発のモデル化を行った．まず，ある典型的な分散開発の例を調べ，チーム内の作業とチーム間の作業に分けてそれぞれをモデル化した．前者では，Marc Kellner の「ソフトウェアプロセスモデリングのための例題」[1] を拡張してモデル化に用い，分散されたチーム内での作業の概要を示した．後者については，新たにモデルを提案した．このモデルでは，各チーム間でやり取りされる連絡に注目し，連絡の手順，送り手・受け手の状態，属性などを記述する．このモデルから，まだ，完了していない処理の発見などが行える．

Process and Communication Models for Distributed Software Development

Yoshihiko Okada[1], Hajimu Iida[1], Katsuro Inoue[1], Koji Torii[1],
Wataru Nagaoka[2], Hajimu Umemoto[2], and Mutsuo Sakai[3],
[1]Faculty of Engineering Science, Osaka University.
[2]Hitachi Seibu Software Co., Ltd. [3]Hitachi, Ltd.

We have investigated a distributed software development, and have modeled it. In this paper, we will show two models. One is for the processes inside a single team, concerning no interaction with other teams. For this purpose, we have extended our model for Marc Kellner's "Software Process Modeling Example Problem". Another is for the communications between two distributed teams. The model is composed of communication processes, the states of sender and receiver, and attributes. Using this model, we can detect the incomplete communication and other characteristics.

1 はじめに

近年，ソフトウェア開発はその大規模化に伴い多人数による開発を余儀なくされるようになりつつある．そして開発者の増加によって一か所での集中した開発が困難になり，分散した多くの地域での開発が一般的になりつつある．しかし，その大規模化，分散化に付随した多くの問題が生じることにもなった．そのため，ソフトウェアの分散開発についての研究も行なわれるようになってきたが，まだその確立された手法やモデルはあまり提案されていない．

ソフトウェアの分散開発をモデル化することにより，これまで曖昧だった開発の現状や問題を明らかにすることは重要な課題である．また，モデル化することで多くの人々に対して，ソフトウェア分散開発の実情，手法などを容易に伝えることができる．あるいは，そのモデルを枠組として分散開発の支援を実際に行なうといったことも可能となるであろう．

本研究では，まず，ソフトウェア分散開発の現状を探り，二つの部分についてモデル化を行なった．一つは分散開発における各チーム内での作業をモデル化することであり，もう一つはソフトウェア分散開発において特に重要であると考えられる各チーム間でのコミュニケーションにおける用件に着目したモデル化である．前者については，Marc Kellner の「ソフトウェアプロセスモデリングのための例題」[1] について，以前，我々が行なったモデル化を基に拡張することで行なう．後者についてはここで新たに提案する．このモデルでは，これまで曖昧であった連絡の経路を明確にするものである．

2 ソフトウェア分散開発の現状

ソフトウェア分散開発の現状を知るために，ある地方銀行のオンライン業務ソフトウェアを新規に開発するプロジェクトについて調査を行なった．以下はその概要である．

2.1 分散の行なわれている工程とプロジェクトの分割

ソフトウェア分散開発といっても，現在行なわれているプロジェクトの多くは，すべての工程が分散して行なわれているわけではなく，今回調査したプロジェクトでも図１に示すようにある部分についてのみ分散して行なわれているのが現状である．

システム計画とシステム設計工程の前半には，小人数で一箇所に集まって作業が行なわれている．システム計画時に行なわれる作業は，顧客の要求を分析して基本仕様書を作成することであり，システム設計で行なわれる作業は，基本仕様書を機能別に分割してそれぞれを詳細に記述した機能仕様書を作成する作業である．

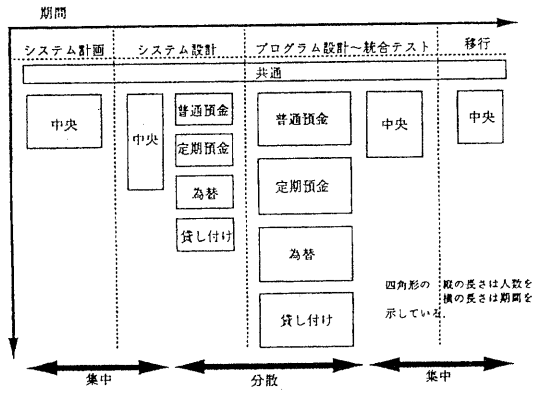


図 1: プロジェクト例の構成

この段階は，複数人で互いに意見を交換しながら作業を進めるという点や，あるいは，まだ仕様もはっきり決まっておらず，作業をうまく分割できないといった理由から，分散させることが困難であると考えられる．また，移行前の統合テストなども実際と同一の環境で一箇所に集まって行なわなければならないことが多く，通常，分散できない（一箇所に集まって作業する場合の小人数のチームを，ここでは中央と呼んでいる）．

従って，システム設計途中（機能の分割が終了した段階）から，最終の統合テストの前までが分散して行なわれている．そしてこの段階がもっとも関わる人数が多い．

プロジェクトの分割は業務単位などで行なわれることが多く，このプロジェクトの場合には，大きく普通預金，定期預金，為替，貸し付けの４つの部分をそれぞれ開発するサブプロジェクトに分割されていた．また分割されたサブプロジェクトはその内部でも，いくつかの小さなサブプロジェクトへと分割されている．

このようにプロジェクトは分割され，各チームに割り当てられる．この場合のチームは東京と大阪など遠隔地に分散していたり，同一会社内だけでなく外注などで他会社に割り当てられることもある．チームが多人数のときはさらに小チームへと分割される．１チームは１０～１５人程度で構成され，それぞれリーダーが存在する．これ以上人数が増えると管理が行き届かなくなり，これより少ない場合はコスト面で問題がある．リーダーは一般のプログラマより経験の豊富な者が務める．また，業務によって分割されたチーム以外に，それぞれのチームで使用される共通部品を作成したり，環境の整備などを行なう共通チームも存在している．

2.2 チーム内で行なわれる作業

システム設計の後半から，最終統合テストの前までの工程において各チームで行なわれている作業は，次のようなものがある．

・ソフトウェア設計書の作成
・ソースコードの作成
・テスト環境の作成
・テスト

また，これ以外の仕事として，

・スケジューリングや進捗管理
・チームの外部との連絡やそれに対応する処理

などが行なわれる必要がある．これらの仕事を適宜チーム内の各員に割り当てる．

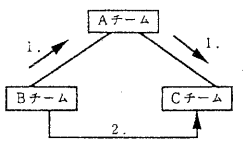
2.3 チーム間でのコミュニケーション

チームに分かれて作業を行なう場合，先に述べたチーム外部との連絡や情報の管理ということが重要になってくる．現状ではチーム間での連絡は次のような方法で行なわれている．

・リーダー会議
・レビュー
・連絡票

リーダー会議やレビューは一箇所に集まって行なわれる．リーダー会議は定期的に開かれ，進捗や開発中に生じた問題点などについて話し合われる．また，レビューはソフトウェア設計書作成の途中で必要に応じて行なわれている．また，一般の連絡は連絡票によって行なわれる．形式は決まっており，社内便やＦＡＸによってやりとりする．

連絡票のやりとりは図２のように行なわれる．連絡票の受渡しなどが上位のチームや管理者を通さずに受け渡されていることもある．



1．上位チームを通して連絡を行う場合
2．下位チーム同士で連絡を行う場合

図 2: 連絡表のやりとりの例

2.4 コミュニケーションの問題

ソフトウェア分散開発において特に重要であると考えられるコミュニケーションの問題には次のようなものがある．

連絡事項が完了しないまま，放置される．

分散したチーム間でのコミュニケーションは不完全なものになりがちで，連絡事項が完結しないまま放置されるようなこともある．連絡が来ていても読まなかったり，気がつかなかったりする．急いでいて，あとで了解をとるつもりだったのを忘れてしまうということもあり得る．

関連した連絡事項が分からなくなる．

連絡の整理が不十分であることから，時間の経過などによって関連事項が分からなくなる．

連絡する場合の経路が曖昧である．

勝手に下位のチーム同士で連絡を取りあって，その結果を上位のチームに連絡していないことがある．

チーム内で非公式な情報が飛び交っている．

仕様の曖昧さに対し，非公式につじつま合わせなどを行なっている．例えば，隣の人間が書いている仕様書を覗いてみて気がついて直すというようなことや，他の人と話しをしていてふと間違いに気がついたりすることなどがある．また，さまざまな作成基準にも曖昧さがかなり含まれているため，基準通りに作っても人によって異なるものができたりする．このような場合の調整はチーム内では容易に行なえるが，分散したチーム同士では困難である．

このようにチームの中では，インフォーマルな情報がかなり飛び交っていて，その中に他チームにとっても重要な情報が含まれている場合が少なくない．

記録が不完全である．

個人が独自の解釈をしたまま，記録を残さずに仕事を続けており，後で問題になることがある．

チーム同士での体制が異なる．

あるチームが他の会社にあるような場合には，その会社の体制ややり方，文化などの違いから，チーム間でのギャップがより大きくなる．しかも，他の会社のやり方に干渉することは困難である．

チーム間の連絡を促進するような基盤技術がまだ十分でない．

電子メールなどによる連絡や，構文エディタ等による仕様作成支援もまだ普及していない．

このような問題のいくつかの改善に役立つよう分散開発のモデル化について考える．

3 ソフトウェア分散開発のモデル化

ソフトウェア分散開発のモデル化を行なうためにここでは,

1 各チーム内での開発作業

2 各チーム間の連絡のインターフェースとその処理

の二つに分けてそれぞれをモデル化する. 1では, Kellner の例題をもとにしたモデルを流用し, 2についてはここで新たに用件の状態遷移に基づくモデルを提案する.

3.1 チーム内における作業のモデル化

3.1.1 Kellner の例題

近年, ソフトウェア開発過程を形式的に記述する試みがなされているが, Marc Kellner の「ソフトウェアプロセスモデリングのための例題」は共通の問題をさまざまな方法でモデル化・記述することによりそれぞれの方法についての理解や把握および比較を助けるという目的で提案された. 現在この問題に対してさまざまな記述が実際に行なわれている [2][3].

Kellner の例題はソフトウェアの変更作業の一部を英文十数ページにわたって規定したもので, 核問題といくつかの拡張問題とからなる. 核問題では一つのモジュールの変更作業のみを扱っている. 問題全体が規定する作業には Develop Change and Test Unit という名前がつけられており, 以下の8つのサブステップに分けられている.

(1) **Schedule and Assign Tasks** (タスクのスケジューリングおよび割り当て) ソフトウェアの変更に関する作業のスケジュールを作成し, 個々のタスクをメンバに割り当てる.

(2) **Modify Design** (デザイン変更) 要求変更により影響を受ける単体コードユニットのデザインの修正を行なう.

(3) **Review Design** (デザインレビュー) 修正されたデザインのレビューを行なう. 結果によってはデザインの変更のステップをやり直す.

(4) **Modify Code** (コード変更) デザインの変更をコードに実現し, コンパイルを行ないオブジェクトコードを生成する.

(5) **Modify Test Plan** (試験計画の変更) 要求変更に関する機能の試験を行なうための試験計画の変更を行なう.

(6) **Modify Unit Test Package** (単体試験パッケージの変更) 試験計画の変更に従って, 試験パッケージの変更を行なう.

(7) **Test Unit** (単体試験) 修正されたコードについて試験パッケージを実行し, その結果の解析を行なう. 試験が不合格である場合は, ソースコードの修正, 単体試験の修正の一方または両方を行なった後, 再試験を行なう.

(8) **Monitor Progress** (進捗状況管理) 上記の各ステップの進捗状況を管理する. スケジュールからの逸脱が深刻な場合には, 再スケジューリングを行なう.

各ステップにはそれぞれ様々な開始・終了条件などが付加されている. ステップ間で受け渡されるプロダクトやそれぞれの担当者などについても詳細に記述されている.

各ステップ間の関係の概略を図3に示す.

モジュールの変更の要求を受けとるとプロジェクトマネージャ (リーダー) がスケジューリングを行なう. 仕事の割り当てが完了すると, 実行可能なステップはそれぞれ並列に実行することができる. 最終的には, 新しいコードがテストに合格し, その通知を進捗状況管理が受けとるとすべてのステップが終了する.

このように Kellner の例題は比較的現実的で, 扱うメッセージや関わる人間についての規定もなされている. 我々はこの例題に対するモデル化をすでに行なっており [4], 今回は各チーム内部での作業については, すでに作成したモデルを流用・拡張して用いることにする.

3.1.2 Kellner の問題の拡張

Kellner の例題の核問題では, 単体モジュールの変更だけを扱っているため, これを複数モジュールの開発を扱うものに拡張する (これは, Kellner の例題の拡張問題の一つに当たる. Kellner の例題では, そのアプローチの能力を示すために, さまざまな拡張が許されており, その解釈はそれぞれに任されている).

ここでは, 以下のように拡張を行なうことにした. モデルの最下層には, 核問題の記述をそのまま当てはめる. その上位に当たる層として, 下位の層を統合する作業が要求される. そこでは, ソースコード自体は作成しないが, 統合テストと, そのためのテストパッケージの作成が行なわれる. これらのステップのタスク割り当てやスケジューリング, 進捗状況管理も必要となる.

実際のプロジェクトにこれを当てはめると図4のように拡張される.

3.2 インターフェースと連絡処理のモデル化

ここでは, チーム間インターフェースと連絡処理のモデル化について述べる. チーム間でやりとりされ

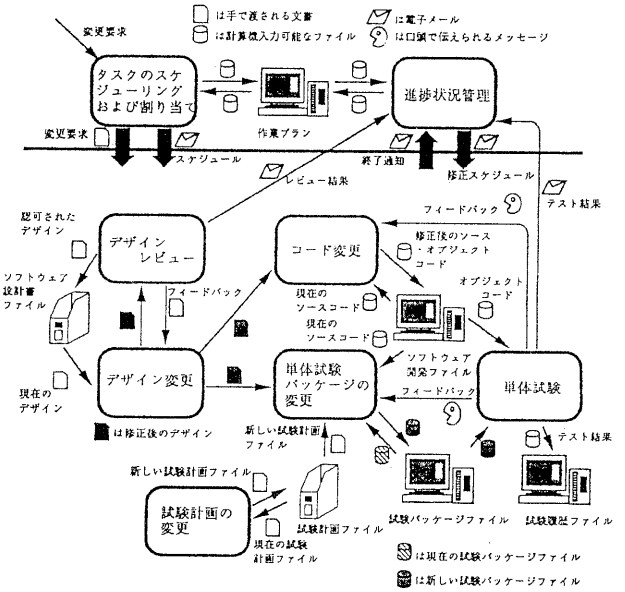


図 3: 核問題における各ステップ間の関係

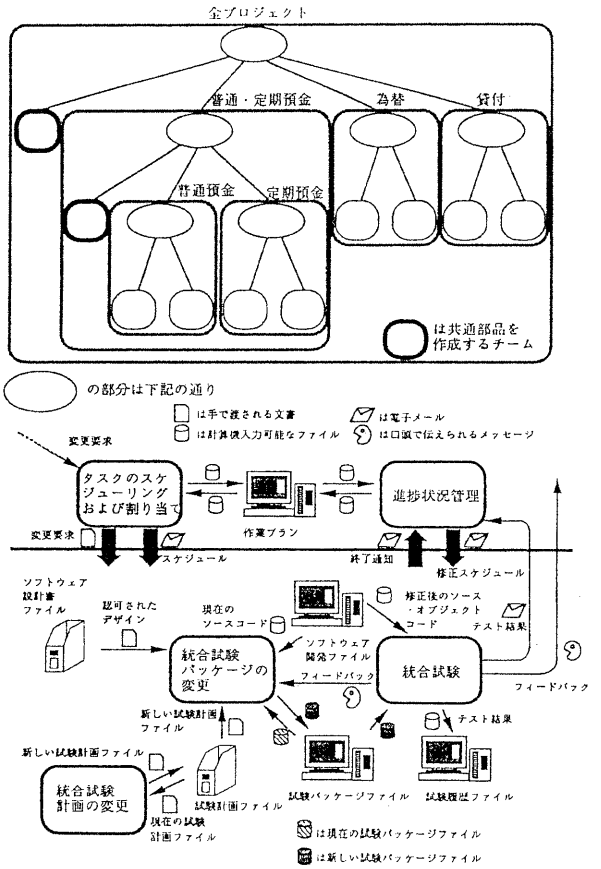


図 4: 分散開発のための拡張モデル

る連絡にははそれぞ目的・案件などという，対象となる主題を持っている (この主題のことを以下では単に用件と呼ぶ)．つまり，一つの用件に対する処理としてさまざまな連絡がやりとりされる．

ここでは，チーム間のインターフェースを扱う部分が複数の用件を管理し，それぞれの用件が処理済みとして完了するまで，それらに関わる連絡を入出力としてとり扱うものとしてモデル化する．

3.2.1 処理の内容

チーム間のインタフェースの部分では，各用件に対して具体的には表1に示すような処理が行なわれている．

表 1: 処理内容の例

用件の分析
用件に対する回答の作成
チーム内部での調査・作業
用件に関連する部署の決定
上位チームへの調査・作業依頼
下位チームへの調査・作業依頼
途中経過報告の作成
など

このような連絡処理を行う場合，その手順は図5のような構造を持つと考えられる．

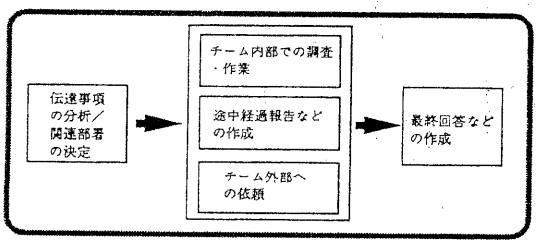


図 5: 用件処理の手順

このとき伝達される現実のプロダクトとしては表2に示されるようなものがある．

3.2.2 連絡経路の管理

連絡経路は，連絡内容の扱う用件とその処理内容に従って管理される．

例えば，図6のチームCでコードを変更した場合，その内容をチームBに送り，Bではその変更で問題ないかどうかを調べ，あるいは，他に関連のありそ

表 2: 伝達されるプロダクトの例

質問票
回答報告書
ソースコード
仕様書
ノウハウ
スケジュール
タスク割り当て
テストパッケージ
など

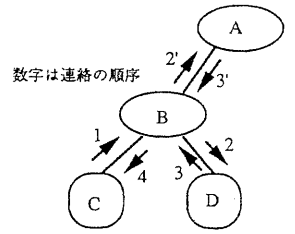


図 6: コード変更通知の連絡経路

うな部所としてチームAやDがあれば，変更の内容をそれぞれに送る．A，Dでも同様の調査を行ないBに回答を返す．Bでは，それらの回答と自チーム内での調査結果を総合してCへ回答を送る．

以上のようなことが行なわれているとき，用件の種類とそれに対応した伝達経路を明確にすることで，次のような点が改善されると思われる．

- 用件の完了を明確にする．
- 連絡の窓口を一箇所にし，下位チーム同士での勝手なやりとりをさせない．
- 連絡の記録を残し，関連するものを追跡できる．

以下では，この経路を明確にするため用件のもつ
・状態
・属性
・操作
について述べる．

3.2.3 用件の状態

ここでは，チーム間におけるコミュニケーションの基本構成単位を message と呼ぶことにする．このとき，ある用件に関して，例えば，図7のような message の組が存在すると考えられる．

送り手が何らかの要求として message 1 を送った時，受け手ではまず，これを受けとったという確認

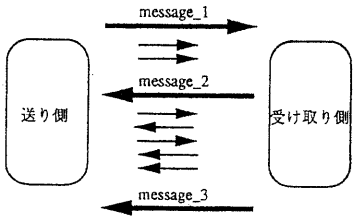


図 7: 用件を表す message の集合

として message 2 を返す．さらに message 1 の最終的な回答として message 3 を返す．また，1 と 2 の間では，送り手から受け手への受け取りの確認を催促する message があり，2 と 3 の間には，送り手から受け手への仕事を急がせる催促や，受け手から送り手へは仕事の途中経過の報告などがあると考えられる．

1 から 3 までの全ての message が 1 つの用件を表すとものとし，ある伝達事項に対して図 8 のような状態遷移を考える．

ある用件に関連して派生する用件もあるため，モデルは階層的に構成される．もとの問題から派生した用件（子の用件）全てが完了すると，もとの用件（親の用件）も完了する．最上位の用件が完了することで，用件全体の完了となる．

3.2.4 用件の属性

用件がもつ属性としては次のようなものが考えられよう．

type 要求・質問（返答を必要とするもの），配布・通知（返答を必要としないもの）．

content ソースコード，ドキュメント，スケジュール，ノウハウ など．

check 用件について調べて，問題がなければ次のチームに渡し，問題がある場合には次のチームには渡さないなどのチェックをする必要があるかどうか．チェックしない場合や，急ぐ場合などとりあえず次のチームに渡しておいて後でチェックするなどを示す．

from 差出人・差出チームなど．

to 受け取りの宛先，全プロジェクトへのブロードキャスト，ローカルなブロードキャストなど．

limit 期限，緊急度など．

parent 親の（元の，そこからこの用件が派生した）用件．

child 子の（新たに派生した）用件．

3.2.5 用件に対する操作

用件に対して行なわれる操作や連絡経路などは用件の属性に基づいて決定される．図 9 は共通部品を扱うチーム B から部品の仕様変更許可願いを連絡する場合である．この場合，連絡は A にまず送られるが，変更の影響をうける C や D にも送る必要がある．しかも，変更されると問題がある場合には変更を許可しないなどの決定をしなければならないため，それぞれのチームでは問題の有無についてチェックを必要とする．このような場合に，差出人が共通部品を扱うチームである時は，チェックを必要とするローカルブロードキャストであるというようなルールを定めておけば，属性から連絡経路などを決定することができる．

4 おわりに

ソフトウェア分散開発の実例をもとに，チーム内で行なわれている作業とチーム間のコミュニケーションという二つの部分に対してモデル化を試みた．

前者では，Kellner の例題を流用して分散されたチーム内での作業の概要を示した．これをもとにして，チーム内の各個人に対しての開発支援などを行なうことができる．

後者では，チーム間でやりとりされる用件の状態・属性・操作について着目し，連絡経路を明確にするためのモデルを提案した．このモデルをもとに，完了していない用件の発見や，関連のある用件の追跡などが容易に行なえるような，チーム間の連絡支援ツールの開発が可能だと考えられる．また，このモデルから連絡のためのコストを割出すことについても，今後，検討を行ないたい．

その他，今回述べたモデルにチーム内部での非形式的なコミュニケーションについてのモデルなどの別のモデルを加えることについても考えていきたい．

References

[1] Marc Kellner et al.: “ISPW-6 Software Process Example”, Proceedings of the 1st International Conference on the Software Process, pp.176-186, (Octorber 21-22, 1991).

[2] “Proceedings of the 1st International Conference on the Software Process”, (Octorber 21-22, 1991).

[3] “Proceedings of the 7th International Software Process Workshop”, (Octorber 15-18, 1991).

[4] 岡田 他: “「Kellner のソフトウェアプロセス問題」の記述の試み”, 情報処理学会第 43 回全国大会, 7J-1, (1991-10).

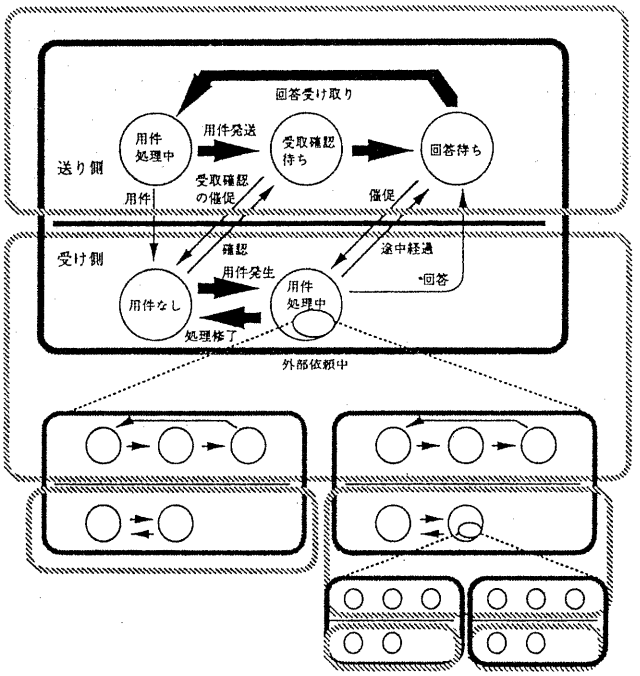


図 8: 用件に対する状態遷移

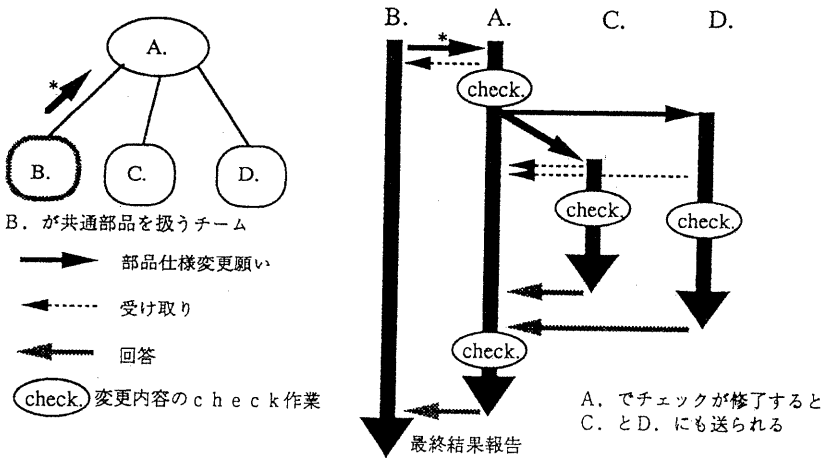


図 9: 用件の伝達と処理の例