ソフトウェア工学 83－15
（１９９２．２．７）

ソフトウェア協調型設計過程のモデル化と知的支援方法について

古宮誠一

情報処理振興事業協会　技術センター

ソフトウェア分散開発を可能にするには、データ伝送機能や伝送上のセキュリティ機能だけではなく、分散開発環境において、協調的に進められるソフトウェア設計過程を強力に支援する必要がある。本稿では、このようなソフトウェア設計過程を協調型設計過程と呼び、これを強力に支援する方式を提案している。

キーワード：ソフトウェア協調型設計過程，ソフトウェア分散開発，ＡＴＭＳ，設計上の意思決定と根拠の再利用，事例ベース推論，半順序集合

Software Collaborated Design Process Modelling and Its Intelligent Support Method

Seiichi KOMIYA

Software Technology Center
Informatin-Technology Promotion Agency, Japan (I.P.A.)

6F Shuwashibakoen 3-chome BLDG.
1-38, Shibakoen 3-chome, Minato-ku, Tokyo 105, Japan

In order to realize software distributed development, facilities to support software design process which is performed cooperatively are desired as well as data transmission facilities with seculity functions. Author calls the above software design process `collaborated design process', and proposes a method to support this process powerfully.

Key Words: Software Collaborated Design Process, Software Distributed Development, ATMS, Reuse of Design Decision and Design Rationale, CBR, Partially Ordered Set

1．はじめに

ソフトウェアの大規模化・複雑化に伴い、作業場所を一ヶ所に集中して開発することが困難になってきた。このため、一つのソフトウェア・システムを複数の作業場所の作業者が共同で開発する形態（＝分散開発）を採らざるを得なくなっている。しかし、その割に分散開発が浸透していないのは、支援ツールなしでは分散開発を実施できないからである。分散開発を可能にするには、分散開発に必要な情報、特に作業の中間成果物などの大量のデータを遠く離れた作業場所に安全かつ確実に伝送するツールが不可欠である。しかし、それだけでは充分ではない。何故なら、そもそもソフトウェアの開発作業というものは、複数の作業者間で協調して進めるべきものであり、しかも、分散開発においては、協調すべき作業者の作業場所が互いに遠く離れているからである。従って、これらの作業者間での協調過程を強力に支援するツールが必要である。従って、ソフトウェア分散開発を可能にするツールには、データ伝送機能、伝送上のセキュリティ機能、分散開発環境での協調過程支援機能の３つが必要である。

ところで、ソフトウェア開発工程から見れば、下流工程では複数の作業がただ並行して進められればよく（＝協調過程支援機能の必要性は低く、ときには皆無でもよい）、大容量のデータ伝送機能が必要となる。逆に、上流工程では小容量のデータ伝送機能でもよいが、強力な協調過程支援機能が必要となる。故に、前者を指向した開発支援環境による開発形態を分散並行開発と呼び、後者を指向した開発支援環境による開発形態を分散協調開発と呼ぶ。分散並行開発を支援する環境としては、ICAROS [1]のなどの研究があるが、分散協調開発を支援する環境の研究は殆どない。従って、ソフトウェア分散協調開発、即ち、分散開発環境でのソフトウェア設計作業を協調して進める過程を知的に支援する技術の開発が望まれる。

2．ソフトウェア協調型設計過程と協調システムのモデル

ソフトウェアの開発では、開発対象の大規模化・複雑化により多人数で開発することが普通となっている。このような状況での開発作業は、複数の作業者間で協調して進められなければならない。特に、ソフトウェア開発の上流工程では、設計上の意思決定のために、その設計に関係する作業者間での合意が必要であり、その作業は協調して進められなければならない。このように協調的に進められるソフトウェア設計過程をソフトウェア協調型設計過程（software collaborated design process）と呼ぶ。

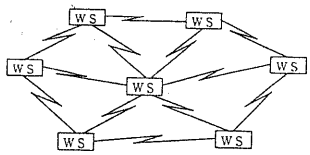

図１-a　ネットワーク状のシステム構成
（管理者なしのモデル）

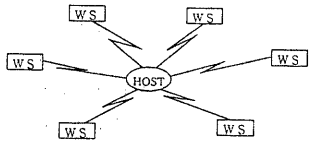

図１-b　放射状のシステム構成
（集中型制御のモデル）

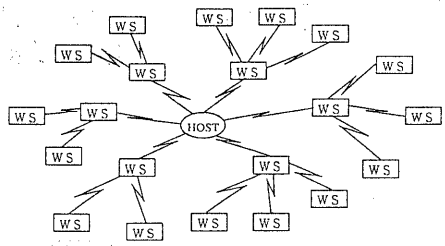

図１-c　木構造状のシステム構成
（階層型制御のモデル）

ソフトウェア協調型設計過程は、協調システムと見なすことができる。協調システムのモデルは、協調のための制御の在り方から次の３つに分類できる。１つは、マネージャの存在を仮定しない（＝フラットな）モデルであり、システムの形態が図１-aのようなネットワーク（＝網）状のシステム構成になるという特徴がある。これを管理者なし（または平面型）のモデルと呼ぶ。残りは、マネージャの存在を仮定するもので、マネージャの人数によってさらに２つに分けられる。１つは、マネージャが一人だけのモデルであり、のようなシステムの形態が図１-b放射状のシステム構成になるという特徴がある。この場合には、制御が一人のマネージャに集中するので、これを集中型制御のモデルと呼ぶ。もう１つは、複数のマネージャが階層的に配置されたモデルであり、システムの形態が図１-cのような木構造状のシステム構成になるという特徴があ

る。これを階層型制御のモデルと呼ぶ。なお、分散型制御のモデルという言葉は、ここでの管理者なしのモデルだけを指すのか、階層型制御をも含めて指すのか、紛らわしいので用いないほうがよいと考えている。ソフトウェア開発のような知的で複雑な作業を協調して進めるには、管理者なしのモデルでは困難であろう。事実、実際のソフトウェア開発では、管理者の存在を仮定した集中型または階層型のプロジェクトで開発が進められている場合が殆どである。このため、ここでは管理者の存在を仮定するモデルを想定するが、話を簡単にするために、以下では集中型制御のモデルに焦点を絞って考察する。

集中型制御のモデルに基づくソフトウェア協調型設計過程では、チーム・リーダの良い悪いがそのままそのチームの良い悪いを決定することは、我々のよく経験するところである。従って、メンバーの意見を巧く抽き出すように、チーム・リーダの行動を知的に支援するとともに、協調に導くアルゴリズムを与えることによって、メンバー全員の意見を協調に導くシステムを構築することは大きな意味を持つ。

3. 集中型制御モデルに基づくソフトウェア設計過程

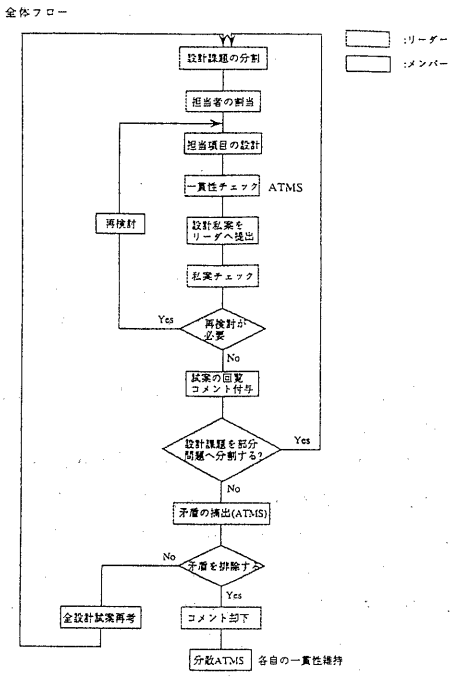

図2　ソフトウェア協調型設計過程での設計手順

議論を明確にするために、本稿では次のような場面を想定する。即ち、複数の作業場所の作業者が、それぞれの席から通信回線を介して、自分の意見や考えなどを交換する形で作業を進めて行く場面を想定する。ソフトウェア開発工程としては、要求仕様が固まった直後の、各要求項目をどのように実現するか（＝設計仕様）を検討する工程を想定する。このような場面では、ソフトウェアの設計作業を協調的に進めなければならないことは明かであろう。このとき、このチームのリーダが、メンバーの意見を抽き出しながら、ソフトウェアの設計仕様をまとめて行く過程を考える。このような過程は、集中型制御モデルに基づくソフトウェア協調型設計過程の典型的な例である。

このとき、ソフトウェア設計仕様の検討ために、設計チームがとる行動を順番に列挙すれば次のとおりである。

◎①設計項目をリストアップする。

◎②設計項目を個別に検討可能な項目と共通の問題として同時に検討すべき項目とに振り分ける。

◎③設計項目を検討すべき順序に並べ直す。

④設計項目別に担当者を割り当てて、各担当者に作業を依頼する。

◎⑤各担当者は自分に割り当てられた設計項目を検討し、設計私案としてリーダに提出する。

⑥リーダは、提出された設計私案の内容をチェックし、一定の水準に達していると判断したときには、これをチームの設計試案としてメンバーに回覧する。設計私案の内容が一定水準に達していなかったときには、コメントを付けてこれを担当者に返却するとともに、再検討を促す。

◎⑦担当者は再検討の後に設計私案をリーダに再提出する。

（再提出の後は⑥に戻る）

⑧メンバーは、回覧されている設計私案に対して意見があれば、コメント票の形で各自の意見を提出する。意見がなければ、その旨をコメント票に明記してこれを提出する。

◎⑨リーダは、入手したコメントをもとに各設計項目ごとに意見調整を行う。このとき、リーダは、採用すべきコメントであれば、これを設計試案の作成者に送りコメントの吸収を促す。却下すべきコメントであれば、これをコメント提出者におくり、コメント取り下げを促す。

（コメントの内容によっては、①－③のいずれか

に戻ることもある。）

◎⑩コメントの却下を指示されたメンバーは、コメントを無条件に取り下げる。

◎⑪コメントの吸収を指示された担当者は、設計試案を再検討し、設計私案としてこれを再提出する。（再提出の後は⑥に戻る）

⑫設計項目全部の検討が終わるまで⑤～⑪を繰り返す。

上記のうちで、コンピュータによる知的支援が可能だと考えられるのは、①②③⑤⑦⑨⑩⑪（いずれも◎が付いている）の８つである。

4．改良型IBISの適用によるソフトウェア設計方法

Conklin によれば、ソフトウェアの保守に要する費用は、ライフサイクル全体での開発コストの約75%にも及ぶという。また、保守に要する費用の約50%は、既存ソフトウェアの理解するための工程「再設計」に要するものだという。一方、Curtisら[１]は、設計時におけるコミュニケーションとコーディネーションの不足がもたらす「ゆゆしき問題」を示した。これらの問題を踏まえて、Conklin らは、設計上の意思決定における議論の状態遷移を構造的に表現するためのモデルIBIS[２]を提案した。IBISでは、設計プロセスを構成する主要な構成要素は、顧客・設計者・インプリメンターなどの間での会話であるとし、これらの間の会話を図３のような Issue－Position－Argumentという枠組みで表現する。これは、設計上の論点（ Issue）に対して、それぞれの専門家からの複数の意見表明（Position）がなされ、各表明に対して賛否の議論（Argument）がそれぞれ展開されるという考え方に立っている。そして、これらの会話による非公式な情報の獲得を支援し、得られた膨大な情報を組織化し検索するためのツールとして gIBIS[]を開発した。

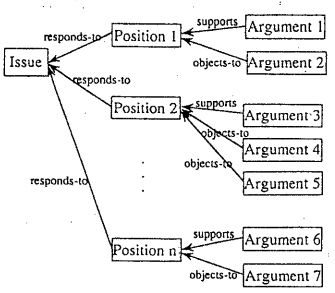

図３　IBISによる対話型（協調型）設計過程

IBISでは、１つの Issueに対して複数の専門家がそれぞれの意見を述べ、意見のそれぞれに対して賛否の議論が複数なされることを前提にしている。即ち、１つの Issueに対して複数のPositionがあり、それぞれのPositionに対して複数のArgumentがあることを前提にしている（図３参照）。このモデルは、１つの論点に対して、複数の専門家がほぼ同時にものを考え、それぞれの立場での直観に基づいて議論を進める形で、問題解決を図る場合には確かに有効である。しかし、このモデルは、 Issueを与えられてからPositionまでに時間を要する場合には適合しない。この場合には、図４のように、１つの Issueに対して、原則として１人の専門家が意見を述べ、この意見に対して複数の人による賛否の議論がなされる。例えば、チームを組んでのソフトウェア開発の場合には、１つの設計課題（ Issue）に対して、原則としてメンバーの１人が割り当てられ、その人が具体的な設計提案（Position）をする。これに対して他のメンバー全員が賛否の意見（Argument）を述べるのが普通であろう。

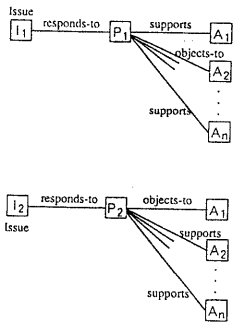

図４　改良型IBISによる協調型設計過程
（Issue1とIssue2が並行検討可能な場合）

この場合においても、１つの設計課題に対して、複数の設計提案がなされる場合がある。それは、１つの設計提案に対して、メンバー全員による賛否の議論だけでは設計課題が解決しない場合である。そのような場合は２つある。１つは対案が必要となる場合であり、もう１つは、部分問題に分割しないと、先に提案された設計案に対する賛否の議論ができない場合である。前者の場合には、１つの設計課題（ Issue）に対して、複数の設計提案（Position）が提出されることになるので、IBIS本来の適用方法と同じになる。後者の場合には、図５のように IssueをSubissueに分解する過程

を明示的に採り入れるが必要があるという点で、IBISの適用方法とは異なってくる。従って、この場合にはハイパーカードにおいて、 IssueとSubissueとを繋ぐリンク・ラベルも必要となってくる（図５参照）。

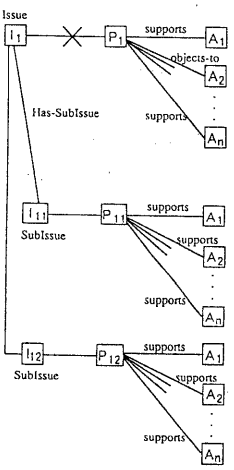

図５　改良型IBISによる協調型設計過程
（IssueがSubissueに分解される場合）

IBISにおいては、IssueとIssueとの間にはPosition提出のための順序関係という概念は存在しない。従って、図４のように、２つの設計課題（ Issue）をそれぞれ個別に検討（即ち、並行検討）可能な場合だけを扱っている場合には問題はない。ところが、ソフトウェアの設計課題と設計課題の間には、図６のように、それらを１つのものとしてまとめなければ設計できないとか、図７のように、一方の設計が終わらなければもう一方の設計を始めることができない、というような作業の順序関係がある。このように、ソフトウェア

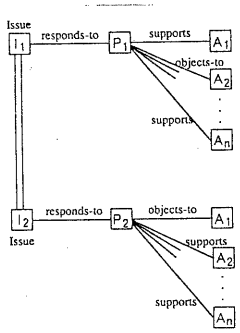

図６　改良型IBISによる協調型設計過程
（複数の課題を１つの課題として検討すべき場合）

の設計おいては、IssueとIssueの間における作業の順序関係という概念が必須となるので、IBISにこれを追加する。従って、IssueとIssueの間、 IssueとSubissueの間、SubissueとSubissueの間に設計作業の順序関係を表すリンク・ラベルが必要となる。

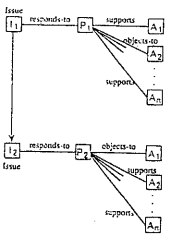

図７　改良型IBISによる協調型設計過程
（Issue1をIssue2よりも先に検討すべき場合）

５．ATMS[2]と分散ATMSによる合意形成過程への支援

本節では、３章で列挙した、設計チームがとる行動１２項目の中の項目５と項目７（以後、これらを設計私案作成過程と総称する）に対する支援を考える。

ATMS(Assumption-based Truth Maintenance System)は、信念の一貫性を維持するシステムである。本章では、入手した設計試案とそれへのコメントを基に、リーダがメンバーを合意へ導く過程の計算モデルとして、ATMSおよび分散ATMSが適用可能であることを示す。そのためには、システムが動作するイメージを示す中で、ATMSおよび分散ATMSで使われているアルゴリズムが、メンバーの意見を合意へと導くための協調計算アルゴリズムとして適用可能であることを示せばよい。

(１)リーダ１人の指示の下に作業を進めるｎ人のメンバー$M_1, M_2, \cdots, M_n$からなるソフトウェア設計チームがあり、リーダおよび各メンバーにはそれぞれ１台のワークステーションが割り当てられている。なお、リーダは管理者専任であってもよく、同時に担当者としての役割を兼務していてもよい。

(２)リーダおよびメンバーからの意見提出はすべて通信回線介して行われる。

(３)各メンバーは、自分に割り当てられた設計項目の設計を自分のワークステーション上で行い、得られた結果（＝設計私案）をリーダに提出する。

(４)各メンバーは意見提出の際、提出者は自分が「真」か「偽」か、いずれの意見を提出しているのか明らかにしなければならない。提出される意見には、すべて

提出者名と提出順番を示す番号からなるＩＤが自動的に付与される。

(5)リーダは、提出された設計私案の内容をチェックし、一定の水準に達していると判断したときには、これをチームの設計試案としてメンバーに回覧する。設計私案の内容が一定水準に達していなかったときには、コメントを付けて作成者に返送するとともに、再検討を促す。

(6)各メンバーは、回覧されている設計試案に対して、自分の意見をコメント票の形でまとめ、設計試案に添付してメンバー全員に回覧する。このとき、コメント票の作成に際しては必ずYes/No/unknownのいずれかを明らかにしなければならない。もし、部分的にYes/No/unknownがある場合には、設計試案を幾つかの部分問題へと分解して、それぞれに対してYes/No/unknownを明らかにしなければならない。

このとき、設計項目のそれぞれに対して寄せる各メンバーからの設計試案およびそれへのコメントのそれぞれは、各メンバーの持つ信念だと見なすことができる。同様に、各メンバーから寄せられる設計試案とそれに対するコメントの集合をリーダが調整して行く過程は、チームとしての信念の一貫性を維持する過程だと見なすことができる。故に、チームとしての信念の一貫性を維持するシステムとしてATMSが適用可能である。これをformalに記述すると次のようになる。

m個の設計項目に対してメンバーＭj（j = 1,2,…,n）が持つ信念の集合を｛$B_{1j}, B_{2j}, B_{3j}, \cdots, B_{mj}$｝とする。このとき、ATMS（= リーダ）は、各メンバーから送られてくるi番目（i = 1,2,…,m）の設計項目に対する、チームとしての信念の集合｛$B_{i1}, B_{i2}, B_{i3}, \cdots, B_{im}$｝の一貫性を維持するために、矛盾の原因となる信念$B_{ij}$をチームとしての信念の集合から追い出すように振舞う。

(7)リーダは、チームとしての信念の一貫性を維持するためのシステムATMSを持っている。

(8)リーダは、１つの設計試案に対するメンバーからのコメントが複数の部分に分解されてYes/No/unknownが述べられていたら、部分問題への分解の在り方を調整した後にATMSを適用する。

(9)ATMSは、メンバーからの意見（= Yes/No/unknown）を信念の集合と見て、それらが無矛盾となるように振舞う。このとき、それぞれの意見の提出者が誰であるかをシステムが管理しているので、取り除かれた意見の提出者が誰であるかを知ることができる。

(10)信念B_{ij}を取り除いた直後に、システムがユーザに制御を返すので、リーダは、信念B_{ij}をそのまま廃棄すべきか、修正後に再提出すべきかを判断の上、いずれか一方をメンバーＭj に指示することができる。取り除かれた意見を削除するには惜しいとリーダが判断した場合には、メンバーに再考を促すメッセージを提出し、設計私案を再提出させる。取り除かれた意見を削除するのが妥当とリーダが判断した場合には、意見の却下を提出者に通知する。

これに対してメンバーＭj はリーダの指示どおりの行動をとる。従って、もし、リーダの指示が信念B_{ij}の廃棄であれば、メンバーＭj は、自分の信念の集合｛$B_{1j}, B_{2j}, B_{3j}, \cdots, B_{mj}$｝から信念$B_{ij}$を無条件に削除しなければならない。メンバーＭj は、これを契機に自分が持つ信念の集合の一貫性を維持しようと努める。このときのメンバーＭj の振舞いは、分散ATMSの動作そのものである。故に、各メンバーが持つ信念の集合の一貫性を維持するシステムとして、分散ATMSが適用可能である。

(11)メンバーは例外なく、自身の信念の一貫性を維持するためのシステム分散ATMSを持っている。

(12)却下を通知された意見提出者は、その通知を絶対的なものとして却下された意見を取り下げなければならない。この折にも、各メンバーは自身の信念の一貫性を維持させるべく、自身が持つ分散ATMSを作動させる。

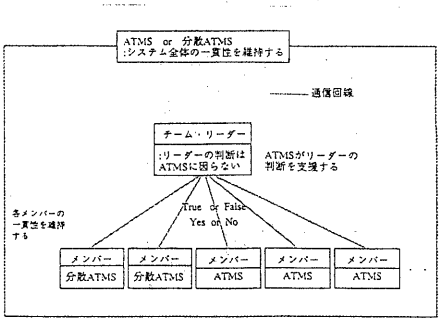

図８　リーダとメンバー間のATMSと分散ATMSの役割

に参考意見を述べてくれるような機能である。このような機能があれば、経験の乏しい者に対してはその経験不足を補い、経験者に対してはその作業の効率化を支援してくれることは明かであろう。このような機能は事例ベース推論による支援機能と呼ばれている。事例ベース推論は、リストアップされた設計項目を個別に検討可能な項目（＝個別検討項目）と共通の問題として同時に検討すべき項目（＝共同検討項目）とに振り分けるときにも適用可能であり、検討項目を検討すべき順序に並べ直すときにも適用可能である。

設計項目抽出過程において事例ベース推論を適用するには、現在検討中の設計課題に対して参考となるような既存の事例をデータベースの中から検索できなければならない。そのためには、話題を特定するためにユーザが与えた1つまたは複数のキーワードを基にその話題に関する所望の事例を検索する機能が必要である。ここで注意すべきことは、その事例を検索するために予め用意したキーワードとユーザが与えるキーワードは必ずしも一致しないということである。これらのキーワードが一致しなければ、事例を検索することはできないので、両者を一致させるために工夫が必要となる。そのような工夫とは「名寄せ」の機能である。名寄せ機能とは、同一の意味を表す同義語として定義されたキーワードであれば、その中のどのキーワードを用いても検索結果が同一になることを保証する機能である。例えば、コンピュータという事項を検索するのに、コンピュータ、電子計算機、電算機、EDPSを同義のキーワードとして定義しておけば、これらの中のどれをキーワードとして用いても、等しくコンピュータという事項を検索するという機能である。

事例ベース推論で次に必要となる機能は、ユーザからの質問に答えるために、検索された事例を基に推論する機能である。設計項目抽出過程においては、話題（＝ここでは設計課題）を特定するためのキーワードと質問項目（＝ここでは、設計項目抽出過程の中のどれに対する質問かを示す識別コード）が与えられたときに、該当する設計課題に関する既存の事例を検索して、その事例における該当項目を回答として呈示する機能である。ここで注意すべきことは、指定された話題に該当する事例は1つとは限らない。このために、事例を特定するためのキーワードが必要になるということである。設計課題の場合には設計意図（＝設計の狙い）などがその候補となる。これは、同一の設計課題でも設計意図によって設計案が変わってくるからである。事例ベース推論では、事例を特定するためのキーワードが必要でであることを、ユーザが初めから気が付いていることは希である。まして、事例を特定するために、どのようなキーワードが必要かを予め承知していることは殆どない。事例ベース推論を適用した結果、それが自分の求めている事例ではない（＝システムによって呈示された参考意見が、自分の設計課題には適用できない）ことが判明したときに、参考意見を呈示する際に使用した事例に関するすべての記述内容を読むことにより、人間は自分の求める事例とその事例との差異が何によって生じたのかを知ることができる。その結果、自分の求める事例を特定するのに必要なキーワードを特定することができる。このようなキーワードは、その話題において、その事例が意味するもの（＝狙い、ここでは設計意図）を表す語彙であることが多い。このようにして、ユーザは必要なキーワードを知り、それを与えることにより、自分の所望する参考意見を得ることができる。この場合にも、キーワードの名寄せ機能は必要である。

6．2　設計私案作成過程への知的支援

本節では、3章で列挙した、設計チームがとる行動12項目の中の項目5と項目7と項目11（以後、これらを設計私案作成過程と総称する）に対する支援を考える。

設計私案作成過程において望まれる支援機能は、その設計課題に対応する設計案を既存の設計事例の中から検索して具体的に呈示する機能である。具体的には、話題（＝ここでは設計課題）を特定するためのキーワードと質問項目（＝ここでは、設計案の内容を呈示せよということ）が与えられたときに、該当する設計課題に関する既存の事例を検索して、その事例における該当項目（＝ここでは設計案の内容）を回答として呈示する機能である。しかし、同一の設計課題でも対応する設計事例は1つだけとは限らない。このため、その設計課題に対応する複数の設計事例の中から、自分の目的に合う設計事例を特定するためのキーワードとして、設計意図、設計する上での前提条件などの項目（＝属性）に対応する値（＝属性値）が使用される。このとき、質問に対する回答として呈示されるのは、設計案の内容（＝設計上の意思決定）とその設計根拠である。もし、呈示された内容が設計者の意図に沿っていれば、ここに設計上の意思決定とその設計根拠の再利用が実現することになる。もし、ここで、その設

計課題よりも先に検討すべき項目（＝先決検討項目）やこれよりも後に検討すべき項目（＝後決検討項目）、共通問題として同時に検討すべき項目（＝共同検討項目）などがあれば、それらも併せて呈示される。

7．おわりに

ソフトウェア分散開発を可能にするには、データ伝送機能や伝送上のセキュリティ機能だけではなく、分散開発環境において、協調的に進められるソフトウェア設計過程（このようなソフトウェア設計過程を協調型設計過程と呼ぶ）を強力に支援する必要がある。

このために、このようなソフトウェア設計過程を協調システムと見なし、そのモデルについて考察した。そこでは、ソフトウェア開発のような知的で複雑な作業にはリーダの存在を仮定したモデルの導入が実際的であり、リーダの行為を知的に支援するシステムの開発が必要であることを主張した。

次に、協調型設計過程におけるソフトウェア設計プロセスを１２のプロセスに分解し、コンピュータによる知的支援が可能な過程と困難な過程とに振り分けた。しかる後に、協調型設計過程の中核となる設計過程を支援するために、 Conklinらが提唱したIBISを改良した協調型のソフトウェア設計方式を提案した。協調型設計過程では合意形成が重要であり、これを支援するためにATMSおよび分散ATMSが適用可能であることを示した。これを強力に支援する方式を提案した。

その他の設計過程への支援のために事例ベース推論が適用可能であることを示すとともに、その適用方法を明らかにした。

現在、ソフトウェア分散協調開発を可能にするための合意形成過程支援アルゴリズムとして、ATMSおよび分散ATMSが適用可能であることを示した。現在、ソフトウェア分散協調開発を可能にするための合意形成過程支援アルゴリズムとして、ATMSおよび分散ATMSが適用可能であり有効であることを実証するために、システムを試作中である。

なお、本稿では述べなかったが、本システムの合意形成過程支援機能と設計上の意思決定とその根拠の再利用支援機能は、特に分散開発環境で開発されたソフトウェアの保守にも有効である。何故なら、ソフトウェアの新規開発のために組織されていたプロジェクトも、ソフトウェア保守の時期には解散され、かつてのメンバーはそれぞれの持ち場に戻っており、そのようなときにこそ、本システムは効力を発揮するからである。

［参考文献］

[1]Aoyama, M.: Distributed Concurrent Development of Software System: An Object-Oriented Process Model, COMPSAC'90, pp.16-30(Nov. 1990).

[2]de Kleer, J.: An Assumption-based TMS, Artificial Intelligence 28, pp.127-162(1986).