(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第5164032号
(P5164032)
(45)発行日　平成25年3月13日(2013.3.13)　(24)登録日　平成24年12月28日(2012.12.28)

(51)Int.Cl.　F I
G06F 9/44 (2006.01)　G06F 9/06 620K

請求項の数 5 (全 28 頁)

(21)出願番号　特願2005-56804(P2005-56804)
(22)出願日　平成17年3月2日(2005.3.2)
(65)公開番号　特開2006-243987(P2006-243987A)
(43)公開日　平成18年9月14日(2006.9.14)
審査請求日　平成20年2月27日(2008.2.27)

(73)特許権者　506301140
公立大学法人会津大学
福島県会津若松市一箕町大字鶴賀字上居合９０番地
(74)代理人　100103333
弁理士　菊池　治
(72)発明者　ニコライ　ミレンコフ
福島県会津若松市一箕町鶴賀　会津大学内
(72)発明者　吉岡　廉太郎
福島県会津若松市一箕町鶴賀　会津大学内
(72)発明者　渡部　有隆
福島県会津若松市一箕町亀賀字村東１７－１１　エステートツルカメパート２　１０３号

最終頁に続く

(54)【発明の名称】編集支援プログラムおよびプログラム編集の支援方法

(57)【特許請求の範囲】
【請求項１】
計算を構成する一連の処理の空間的関係および時間的関係を規定する計算スキームの要素と、計算の各ステップで行なう処理を表す計算式および変数の要素と、計算に用いるデータの入力および計算結果の出力に関する処理の要素と、の少なくとも３個の要素をそれぞれに含む複数の自己説明型コンポーネントからなってコンピュータに計算を実行させる自己説明型プログラムを編集する編集支援プログラムであって、
コンピュータが、前記複数の自己説明型コンポーネントそれぞれについて、前記少なくとも３個の要素それぞれの情報を、マルチメディア言語を用いてディスプレイに表示し、
コンピュータが、前記少なくとも３個の要素それぞれに個別に、それらの要素の変更の入力を受けて、その変更を行い、変更した内容を保存する
ように機能させることを特徴とする編集支援プログラム。
【請求項２】
前記マルチメディア言語は、画像、グラフィックス、アニメーションのうちの少なくとも１種類を用いて表示するものであること、を特徴とする請求項１に記載の編集支援プログラム。
【請求項３】
前記要素ごとの情報は、サイバーフィルムとして構成されていること、を特徴とする請求項２に記載の編集支援プログラム。
【請求項４】

複数のソフトウェアコンポーネントを有して構成されるプログラムの編集を支援するプログラム編集支援方法であって、
前記ソフトウェアコンポーネントそれぞれが、計算を構成する一連の処理の空間的関係および時間的関係を規定する計算スキームの要素と、計算の各ステップで行なう処理を表す計算式および変数の要素と、計算に用いるデータの入力および計算結果の出力に関する処理の要素と、の少なくとも３個の要素を含み、
前記コンピュータが、保存されている複数のソフトウェアコンポーネントから少なくとも一つのソフトウェアコンポーネントを検索して読み込むステップと、
前記コンピュータが、前記読み込みステップで読み込まれたソフトウェアコンポーネントを一連のマルチメディアシーンとして、少なくとも図形および文字を組み合わせてディスプレイに動的に表示する表示ステップであって、前記ソフトウェアコンポーネントの前記少なくとも３個の要素を表示する表示ステップと、
前記コンピュータが、前記少なくとも３個の要素それぞれに個別に、それらの要素の変更の入力を受けて、その変更を行い、変更した内容を保存するステップと、
を実行することを特徴とするプログラム編集の支援方法。

【請求項５】
複数のソフトウェアコンポーネントを有して構成されるプログラムの編集を支援する編集支援プログラムであって、
前記複数のソフトウェアコンポーネントそれぞれが、計算を構成する一連の処理の空間的関係および時間的関係を規定する計算スキームの要素と、計算の各ステップで行なう処理を表す計算式および変数の要素と、計算に用いるデータの入力および計算結果の出力に関する処理の要素と、の少なくとも３個の要素を含み、
コンピュータに、
メモリに保存されている複数のソフトウェアコンポーネントから少なくとも一つのソフトウェアコンポーネントを検索して読み込むステップを実行させ、
前記読み込みステップで読み込まれたソフトウェアコンポーネントを一連のマルチメディアシーンとして、少なくとも図形および文字を組み合わせてディスプレイに動的に表示する表示ステップであって、前記ソフトウェアコンポーネントの前記少なくとも３個の要素を表示する表示ステップを実行させ、更に、
前記少なくとも３個の要素それぞれに個別に、それらの要素の変更の入力を受けて、その変更を行い、変更した内容を保存するステップを実行させる、
ことを特徴とする編集支援プログラム。

【発明の詳細な説明】
【技術分野】
【０００１】
この発明は、コンピュータを利用するプログラム、プログラム改変方法、改変管理方法およびコンピュータ処理方法に関し、特に、自己説明型コンポーネントモデルによる処理に関する。
【背景技術】
【０００２】
大規模なソフトウェア・システムの開発では、機能を細分化しそれぞれの機能単位でソフトウェアコンポーネントを実装し、それらコンポーネントの組み合わせと連携によってシステム全体を構築するのが一般的である。このような開発では、それぞれのコンポーネントがどのようなデータと機能を持つべきかを規定するコンポーネント構造と、それらのコンポーネントが連携し合うための環境からなる、コンポーネントモデルを定義することが必要である。これまでに様々なコンポーネントモデルが提唱され、利用されてきた。現在、最も広く利用されているソフトウェアのコンポーネントモデルとしてＣＯＲＢＡ、ＣＯＭ＋、Ｅｎｔｅｒｐｒｉｓｅ　Ｊａｖａ　Ｂｅａｎｓ（商標）がある。
【０００３】
上述したようなコンポーネントモデルを再利用するには、そのコンポーネントの機能、

利用方法、内部動作を理解する必要がある。従来のコンポーネントでは、一般に、コンポーネントが提供する機能を利用するのに必要最低限の情報がインターフェイス定義として記録されている。すなわち、コンポーネントが提供する関数の名前とその引数のデータ型が記録されている。その他には、コンポーネントの作成者、作成日、バージョン等の情報も記録されている。

【０００４】

しかし、コンポーネントがどのような機能を持ち、どのような内部処理を行なっているのか、どのようにそれらの機能にアクセスでき、どのように拡張できるかなど、そのコンポーネントを理解、利用または拡張するのに必要な情報は、コンポーネントには含まれず、仕様書やマニュアルとして記載されている。

【０００５】

一方、コンピュータサイエンスにおいて、アルゴリズムを学習するためのツールとして、アルゴリズムのフローチャートの作成によるもの（特許文献１参照）、サイバーフィルムによるもの（非特許文献１～３参照）が知られている。

【特許文献１】特開平８－１１０７５４号公報

【非特許文献１】Yutaka Watanobe, Rentaro Yoshioka, Nikolay N. Mirenkov著、"Self-Explanatory Components to Study Algorithms", THE JOURNAL OF THREE DIMENSIONAL IMAGES 三次元映像のフォーラム, vol.16-No.4, 2002年12月, p.231-236

【非特許文献２】R. Yoshioka, N. Mirenkov著、"Visual computing within environment of self-explanatory components", Soft Computing 7, Springer-Verlag 2002年, p.20-32

【非特許文献３】Rentaro Yoshioka, Nikolay N. Mirenkov, Yuho Tuchida, Yutaka Watanobe著、"Visual Notation of Film Language System", DMS Proceedings of The Eighth International Conference on Distributed Multimedia Systems, 2002年9月, p.648-655

【発明の開示】

【発明が解決しようとする課題】

【０００６】

上述のように、コンポーネントを効率的に再利用するには、そのコンポーネントの機能、利用方法、内部動作をいかに簡単に理解するかが大切である。ところが、コンポーネントの機能や内部処理に関して、どのようにそれらの機能にアクセスでき、どのように拡張できるかなど、そのコンポーネントを理解、利用または拡張するのに必要な情報は、コンポーネントには含まれていない。これは、コンポーネントにそれら、いわゆる「説明書」に当たる情報を記述する仕組みがないからである。すなわち、ソフトウェアを開発するプログラミング環境（ツール）にはこれらの情報が含まれていない。

【０００７】

このため、開発の様々な段階において、説明書やマニュアルの中から、最も適切なものを選択し、必要とする情報を検索するという余計な作業を開発者に強いることになる。初心者にとっては、これら説明書やマニュアルにいかに早く精通するかということが開発の大きな割合を占めるようになる。

【０００８】

これは、現在実用されているコンポーネントモデル全てに共通する問題である。コンポーネントの意味や動作が理解しづらいという問題は、コンポーネントの修正、拡張などの再利用に当たって、開発者（ユーザ）に多大な負担をかけ、ソフトウェアの生産効率を著しく低下させる原因だと考えられる。

【０００９】

本発明の目的は、上記事情に鑑みて、ソフトウェアコンポーネントを理解し利用するのに便利な自己説明型コンポーネントモデルによる自己説明型プログラムを編集する編集支援プログラムおよびプログラム編集の支援方法を提供することにある。

【課題を解決するための手段】

【００１０】
この発明は上記目的を達成するものであって、請求項１に記載の発明は、計算を構成する一連の処理の空間的関係および時間的関係を規定する計算スキームの要素と、計算の各ステップで行なう処理を表す計算式および変数の要素と、計算に用いるデータの入力および計算結果の出力に関する処理の要素と、の少なくとも３個の要素をそれぞれに含む複数の自己説明型コンポーネントからなってコンピュータに計算を実行させる自己説明型プログラムを編集する編集支援プログラムであって、コンピュータが、前記複数の自己説明型コンポーネントそれぞれについて、前記少なくとも３個の要素それぞれの情報を、マルチメディア言語を用いてディスプレイに表示し、コンピュータが、前記少なくとも３個の要素それぞれに個別に、それらの要素の変更の入力を受けて、その変更を行い、変更した内容を保存するように機能させることを特徴とする。

【００１１】
また、請求項２に記載の発明は、請求項１に記載の編集支援プログラムにおいて、前記マルチメディア言語は、画像、グラフィックス、アニメーションのうちの少なくとも１種類を用いて表示するものであること、を特徴とする。

【００１２】
また、請求項３に記載の発明は、請求項２に記載の編集支援プログラムにおいて、前記要素ごとの情報は、サイバーフィルムとして構成されていること、を特徴とする。

【００１５】
また、請求項４に記載の発明は、複数のソフトウェアコンポーネントを有して構成されるプログラムの編集を支援するプログラム編集支援方法であって、前記ソフトウェアコンポーネントそれぞれが、計算を構成する一連の処理の空間的関係および時間的関係を規定する計算スキームの要素と、計算の各ステップで行なう処理を表す計算式および変数の要素と、計算に用いるデータの入力および計算結果の出力に関する処理の要素と、の少なくとも３個の要素を含み、前記コンピュータが、保存されている複数のソフトウェアコンポーネントから少なくとも一つのソフトウェアコンポーネントを検索して読み込むステップと、前記コンピュータが、前記読み込みステップで読み込まれたソフトウェアコンポーネントを一連のマルチメディアシーンとして、少なくとも図形および文字を組み合わせてディスプレイに動的に表示する表示ステップであって、前記ソフトウェアコンポーネントの前記少なくとも３個の要素を表示する表示ステップと、前記コンピュータが、前記少なくとも３個の要素それぞれに個別に、それらの要素の変更の入力を受けて、その変更を行い、変更した内容を保存するステップと、を実行することを特徴とする。

【００１６】
また、請求項５に記載の発明は、複数のソフトウェアコンポーネントを有して構成されるプログラムの編集を支援する編集支援プログラムであって、前記複数のソフトウェアコンポーネントそれぞれが、計算を構成する一連の処理の空間的関係および時間的関係を規定する計算スキームの要素と、計算の各ステップで行なう処理を表す計算式および変数の要素と、計算に用いるデータの入力および計算結果の出力に関する処理の要素と、の少なくとも３個の要素を含み、コンピュータに、メモリに保存されている複数のソフトウェアコンポーネントから少なくとも一つのソフトウェアコンポーネントを検索して読み込むステップを実行させ、前記読み込みステップで読み込まれたソフトウェアコンポーネントを一連のマルチメディアシーンとして、少なくとも図形および文字を組み合わせてディスプレイに動的に表示する表示ステップであって、前記ソフトウェアコンポーネントの前記少なくとも３個の要素を表示する表示ステップを実行させ、更に、前記少なくとも３個の要素それぞれに個別に、それらの要素の変更の入力を受けて、その変更を行い、変更した内容を保存するステップを実行させる、ことを特徴とする。

【発明の効果】

【００１７】
本発明によれば、コンポーネントを理解し利用するのに必要な情報をコンポーネント内に記述できるコンポーネント構造が採用され、それから提供される情報から適切なものを

適時取り出し表示することができ、それぞれの情報項目ごとに最適なマルチメディア表現を利用できる。これにより、各コンポーネントの自己説明性が向上し、コンポーネントの利用（理解および作成）が容易になる。

【発明を実施するための最良の形態】

【００１８】

初めに、本発明に係る自己説明型コンポーネントモデルの概要を説明する。自己説明型コンポーネントモデルは、ソフトウェアの効率的な作成と管理を可能にする、既存のコンポーネントモデルにはみられない自己説明性を追求したソフトウェアのコンポーネント化技術である。コンポーネントの役割、使い方、そして内部アルゴリズムを理解するために必要な情報をコンポーネントに必須の情報として規定することで、自己説明性を向上し、理解と利用を容易にしている。この技術を構成するのは、コンポーネントをアルゴリズムの観点から特徴づけて要素ごとに分類／保管するデータ・モデル、それら要素を的確に表現するためのマルチメディア言語群、そして、データ・モデルとマルチメディア言語群に対する操作を提供する開発環境である（図１参照）。

【００１９】

自己説明型コンポーネントモデルの中核をなす、データ・モデル、マルチメディア言語群、開発環境は相互に深く関わりあっている。開発環境は、ユーザの種類や作業内容に応じてコンポーネントにアクセスするためのインターフェイスを的確に組み立てることができる。なぜなら、ユーザにとって必要な情報は状況によって異なるが、既にデータ・モデルによって要素ごとに分類／管理されているので、特定の情報に容易にアクセスすることができるからである。また、状況に応じて異なるマルチメディア言語を使用することができるので、情報をより適した形で表現することが可能となる。このような特徴から、コンポーネントを様々な角度から捉えることが可能となり、それらを状況に応じて使い分けることで、知識、関心、経験の異なる様々なユーザがそのコンポーネントを理解できる可能性を高めているのである。この意味においてコンポーネントの「自己説明性」は高められるのであり、これが本技術の最大の目的であり特徴である。以下では、データ・モデル、マルチメディア言語群、開発環境について順を追って説明する。

【００２０】

［データ・モデル］

ソフトウェアコンポーネントは、ある機能を実現するための計算手順をコンポーネント化したものである。その計算手順を計算のスキームという観点で特徴づけていくと、「計算のスキーム」、「計算式と変数」および「入出力処理」の３つの要素に分解することができる。

【００２１】

ここでいう「計算のスキーム」とは、計算を構成する一連の処理の空間的関係と時間的関係を意味する。図２にその例を示す。図２（ａ）の例では、４つの処理、Ａ、Ｂ、Ｃ、Ｄが、横一列につながり、Ａ→Ｂ→Ｃ→Ｄという順序で実行されることを表している。つまり、ステップ１～４において、黒く塗りつぶされている処理が実行される。図２（ｂ）の例では、同じ４つの処理が図２（ａ）と同じく横一列に並び、Ｄ→Ｃ→Ｂ→Ａと異なる順序で実行されることを表している。図２（ｃ）の例では、同じ４つの処理がこれまでとは違い格子状に並び、Ａ→Ｂ→Ｃ→Ｄと図２（ａ）と同じ順序で実行されることを表している。このように、どのような計算手順も「計算のスキーム」という要素を持っている。

【００２２】

「計算式と変数」とは、図２の例の場合で言えば、４つの処理、Ａ、Ｂ、Ｃ、Ｄそれぞれで実際に行う処理を表す計算式や変数のことを意味する。このように、どのような計算手順も「計算式と変数」という要素を持っている。

【００２３】

「入出力処理」とは、計算に用いるデータの入力や計算結果の出力に関する処理のことを意味する。これには、計算を始める前に行う初期値の入力、計算結果のファイルへの出力、計算途中で生ずる他コンポーネントとのやりとりなどが含まれる。このように、どの

ような計算手順も「入出力処理」という要素を持っている。

【００２４】

以上、３つをデータ・モデルで規定する要素とする。ただし、データ・モデルの構造上、規定する要素に制限はなく、対象とするコンポーネントの種類によって必要な要素を付け加えることができる。つまり、最低限、上記３つの要素を含む必要がある。自己説明型コンポーネントとは、データ／知識をこれらの要素でもって表現したもののことを言う。

【００２５】

［マルチメディア言語］

コンポーネントを上記の要素ごとにデータ・モデルに記述するときに用いるマルチメディア言語について説明する。前述の通り、各要素に最も適した専用のマルチメディア言語を用いるため、複数のマルチメディア言語が存在する。ここでは、それらの一例を示す。

【００２６】

＜「計算のスキーム」用の言語＞

「計算のスキーム」は、構造体とそれを構成するノードの形、色、点滅によって表現される。図２で示した例のように、処理をノード（図内の○）として表すと、処理の空間的関係をノードとそれらをつなぐ線によって形作られる構造体として表すことができる。この構造体には幾つかの種類があり、処理に応じて適切なものを選んで使うことができる。構造体の例を図３に示す。図３（ａ）は、代表的な構造体を示している。左から一次元格子、二次元格子、木、ピラミッド、グラフの順に並んでいる。図３（ｂ）は、構造体のノードにまた別の構造体を含んでいる様子を示している。この場合はノードの形を○（丸）ではなく□（正方形）で表現する。このように、より複雑な処理の構造を表すためにノードには別の構造体を含めることができる。図３（ｃ）は、同じ構造体を違う見せ方で表現した例である。このように構造的に同じでも、用途に応じてより適切な、より意味を理解しやすい形で表現できる。

【００２７】

図２の例では、各ノードで行われる処理をＡ、Ｂ、Ｃ、Ｄと表記し区別しているが、本来はこれをノードの色で識別する。図４は、各処理を色で表したものである。ただし、灰色は斜線のハッチングによる。色によって処理を区別しているので、図から同じ処理をしているノードが２つ存在し、残り２つはそれぞれ別の処理をしていることが分かる。このように色で区別することでユーザの理解を助けている。図２の例では、ステップごとに図示することでノードの実行順序、すなわち処理の時間的関係を表しているが、本来はノードの点滅の移り変わりによって表現する。つまり、各計算ステップで塗りつぶされているノードが点滅し、順に移り変わるのである。このとき、点滅しているノードが実行され、具体的にはそのノードに定義されている処理（例えば、数式や関数）が実行される。１つのステップで複数のノードが点滅している場合は、それらの処理が同時に行われる（並列処理される）ことを示す。さらに、点滅の種類で実行される処理のおおまかな性質を表現することができる。点滅の種類には図に示すようなものがある。

【００２８】

図５で示した点滅の具体的な使用例を、図６を用いて説明する。まず図６（ａ）は、全体が点滅している（塗りつぶされている）ノードと輪郭だけが点滅しているノードがある。これは、全体が点滅しているノードがその上、右、下、左のノードの値を参照してなんらかの処理を行っていることを示す。図６（ｂ）は、２つの連続したステップを表わしている。ステップ１では、半分だけ点滅したノードが２つある。これらは、ステップ２で実行するノードを選択するための処理を行っている。ステップ２では、ステップ１の処理で選択されたノードが全体の点滅に変わり、なんらかの処理が行われている。図６（ｃ）は、マルチステージネットワーク上の計算を表わしているが、第２ステージのノードがなんらかの信号またはデータを待っている状態を示している。図６（ｄ）は、２つの連続したステップを用いて、どのように構造体が変化するかを表わしている。ステップ１で、構造体が拡張されるかどうかを判定する処理が点滅しているノードで行われている。その結果、ステップ２で構造体が変化したことを示している。

【００２９】
「計算のスキーム」を表わす具体的な例を図７に示す。図７の例では、二次元格子状の空間的関係において、各列において左から右の順になんらかの計算をし、各計算は一つ前の列（左側の列）の値を参照し現在の列の処理を実行している。
【００３０】
また図７の例では、処理が済んだノードをマークし、処理の時間的関係を読み取りやすくするためにグループオブジェクトを用いている。グループオブジェクトを用いることで、共通の意味を持つノードはグループ化され、その様子を可視化することができる。図８にグループオブジェクトの使用例を示す。図８の例では、各ノードになんらかの処理を行い、その結果からグループに属すかを判断していく処理が実行されている。そして、グループに属するノードをグループオブジェクトで表現している。このように、グループオブジェクトを使用すると、グループ化の様子を可視化することができ、ユーザの理解を助けることができる。
【００３１】
これとは別に、各ステップが表わす意味の抽象度を下げ、ユーザの理解を助けるために、構造体の前面及び背面に、任意の図形、絵、アニメーション、文字を付け加えることができる。これを、前景オブジェクトと背景オブジェクトと呼ぶ。図９に前景オブジェクトの使用例を示す。図９（ａ）の例では、構造体の前面に線を引くことで実行中のノードの移り変わりを分かりやすく表現している。図９（ｂ）の例では、構造体の前面にゼブラな図形を付け加えることで実行するノードと実行しないノードの移り変わりを分かりやすく表現している。図１０に背景オブジェクトの使用例を示す。図１０（ａ）の例では、構造体の背景に地形を表した図を置くことで、構造体が意味することを分かりやすく表現している。図１０（ｂ）の例では、構造体の背景に盤の模様を置くことで、構造体が意味することを分かりやすく表現している。図１０（ｃ）の例では、図形と文字を使うことで、構造体が意味することを分かりやすく表現している。このように、前景オブジェクトと背景オブジェクトは構造体が表現しているものをより具体的に可視化し、ユーザの理解を助けている。また、音を用いて処理の流れや内容、特徴を表すこともできる。例えば、おおまかな計算の変わり目で効果音（例えば、チャイムの音）を鳴らすことや点滅の種類（実行される処理の種類）に応じて異なった音を鳴らすことができる。
【００３２】
＜計算式と変数を表す言語＞
計算スキームの各ステップでは点滅しているノードに定義された計算式が実行される。ここではこの計算式を定義する言語について説明する。
【００３３】
この言語は数学で用いられている一般的表記にアイコンや色などを導入し数学特有の高い抽象性をできる限り少なくするという特徴をもつマルチメディア言語である。この言語は数学や物理における計算のように数字、演算子、変数、関数の各要素で構成されている。ただし、この言語では精度を落とさずに数学特有の抽象度を下げ直感的なわかりやすさを追求するために、「文字における工夫」、「変数の参照における工夫」、「関数表記における工夫」などの工夫がなされている。
【００３４】
＜文字における工夫＞
既存のプログラミング言語では使えないギリシャ文字や記号、様々なフォントを用いることで、分野毎に用いられている独特の表記に比べ、より慣れ親しんだ記述を使うことができるようになっている。例えば、物理ではμは静止摩擦係数、κはバネ定数を意味するなど、特別な意味を持った文字をそのまま用いることができる。
【００３５】
＜変数の参照における工夫＞
数字と演算子は一般的な数学の表記とほぼ同じである。変数は構造体とデータ型という二つの属性を持っている。構造体は、前に図３に示したような空間的関係を表すものであ

り、データ型は格納する値が整数、実数など、どの形式かを表すものである。例えば、二次元格子型構造体の変数が宣言された場合、その格子にあるすべてのノードの値を宣言された変数一つによって扱う事ができる。これは、多次元変数内の値を参照するのに従来のインデックスを用いる方法のほかに、空間内の絶対座標や相対的位置を用いて参照することを可能にする。

【００３６】

図１１にその例を示す。図１１の例では、左側に二次元格子構造上の計算スキームの１ステップを示している。そのステップでは、２つのノードが点滅しており（黒く塗りつぶされたノード）、これらのノードでなんらかの計算が行われていることが分かる（説明のため、各ノードに番号が振られている）。さらにこの場合、二つのノードは同じ色で点滅しているため、同じ処理を行うことがわかる。すなわち、これらのノードには同じ式または関数が割り当てられている。その処理の内容が図１１の右側に書かれた数式である。ここで、変数Ｘはこの二次元格子型構造体に割り当てられた二次元格子型構造体型の変数である。

【００３７】

ここで、各計算式や関数における変数の参照（インデックスの指定の方法）について、従来のプログラムと比較する。ここで二次元格子型構造体についての例をあげると、従来のプログラムでは二次元格子型構造体のｉ行ｊ列の値を指し示す場合にＸ［ｉ］［ｊ］などといった、変数に対してある定数や他のカウンター変数などを用いて値を参照する場合が多い。また、現在値からの相対的なインデックスを指定するためには、Ｘ［ｉ－１］［ｊ＋１］といったように、参照したいインデックスを計算によって求めなければならず、慣れていない人にとっては間違いを犯し易い。ここで説明する言語では、この値の指定や参照といったインデックスの指定を、直接的そして視覚的に表現する。

【００３８】

まず、図１１の数式の左辺は「変数Ｘの現在点滅中のノード」という意味を表し、この計算ステップの場合では、７と１１のノードを示す。図１１の数式の右辺では特別なアイコンによるインデックスの指定が用いられている。ここで、右辺の第一項ではインデックスの参照に長方形で囲まれた２つのリンクされた異なった色（灰色（斜線ハッチングで示す）と黒）のノードが用いられている。このアイコンの右側のノード（黒いノード）は現在点滅しているノードを示し、その左隣にリンクされた灰色のノードが参照されるノードを示している。つまり、このアイコンは、「現在点滅しているノードの左隣のノードの値」を指し示す（このような特別なアイコンをステンシルと呼ぶ）。つまり、この数式は「現在点滅しているノードに、現時点滅しているノードの左隣のノードの値に１を足したものを代入する」という意味である。このように、計算のスキームが定義されているので、数式や関数はその性質を利用して自己説明性を高めることができ、直感的（視覚的）にわかりやすく、直接的な参照や指定が可能となる。

【００３９】

図１２にステンシルの他の例を示す。図１２の上段は二次元格子型構造体におけるステンシルの例であり、（ａ）は現在点滅しているノードの左隣、（ｂ）は右隣、（ｃ）は上、（ｄ）は下のノードを示し、（ｅ）～（ｈ）は四つの子供を持つピラミッド型構造体におけるステンシルの例であり、それぞれ灰色の子ノードを示している。

【００４０】

＜関数表記における工夫＞

関数には、Σ、√、ｓｉｎ、ｃｏｓのような一般的な数学関数と構造体の属性を取得したりするためのシステム関数が用意されている。例えば、図１３（ａ）は数学関数の一つである和を計算するための関数であり、変数ＩＵの値について点滅しているノードと隣接する８つのノードを表示されている定数分の重みをつけて和を取る、ということを意味する。具体的には、点滅しているノードを囲む８つのノードの値を足し、それから点滅しているノードの値を４倍したものを引くということである。これをもっとも一般的なプログラミング言語（Ｃ言語）で表すと図１３（ｂ）のようになる。また、図１３（ｃ）は二次

元格子型構造体の大きさを取得するためのシステム関数であり、Ｎ×Ｎの２次元格子構造の縦及び横のサイズであるＮを取得するためのものである。この関数によって実際のノードの数が分かるためノードの数に依存する計算を行うのに便利である。このようにシステム関数は構造体や処理の流れに関するパラメータや状態を取得するのが主な目的である。

【００４１】

以上が、この言語の具体的な特徴である。この言語は、点滅しているノードをインデックスに用いるなど前項で述べた計算のスキーム用の言語と連携して使われるのが一般的である。そこで、この二つの言語の関係を説明するために、ある計算を両言語で記述したものを示す。ここでは連立一次方程式を解くために用いるＬＵ分解という計算を例に挙げて説明する。ＬＵ分解とは、連立一次方程式を行列方程式Ａ＝Ｌ×Ｕに見立て（Ａは正方行列、ｂは既知のベクトル）、未知のベクトルｘを求めるために、行列ＡをＡ＝Ｌ×Ｕとなるような下三角行列Ｌと上三角行列Ｕに分解して考える計算手法である。すなわち、ＬＵ分解とは、このような行列ＬとＵを求めるための計算である。実際に連立方程式を解くにはさらに前進代入と後退代入という計算を行う必要があるが、この例では省略する。

【００４２】

ＬＵ分解を求めるための一般的な計算式は、行列Ｌの要素をα、行列Ｕの要素をβ、行列Ａの要素をａとした場合、次の式のように表される。

【数１】

$$\beta_{ij} = a_{ij} - \sum_{k=1}^{i-1} \alpha_{ik}\beta_{kj} \qquad (i = 1, 2, \ldots, j)$$

$$\alpha_{ij} = \frac{1}{\beta_{jj}}\left(a_{ij} - \sum_{k=1}^{i-1} \alpha_{ik}\beta_{kj}\right) \qquad (i = j+1, j+2, \ldots, N)$$

【００４３】

この式を、行列の大きさがＮ×Ｎ行列の場合に、ｊを１…Ｎの間で変えながら順次計算すればＬ、Ｕを求めることができる。これを既存のプログラミング言語で記述した例が図１４である。

【００４４】

これと同じ計算を、前述の計算スキーム用の言語で記述したものが図１５および図１６である。これはＬＵ分解の各計算ステップを、一連のフレームとして並べたものであり、各フレームの左上にはそのステップの順番を表す番号が書かれている。各フレームは横一列に並んだ３つの２次元格子構造を含み、それぞれ、左から順にＡ、Ｌ、Ｕに対応している。各計算ステップで、なんらかの処理が行われているノードや参照されているノードが点滅している。そして、処理の終わったノードのグループを分かり易くするために、背景オブジェクト（灰色（斜線ハッチング）で塗られた領域）が付加されている。また、処理の流れの方向（パタン）を分かり易くするために、コントロールストラクチャと呼ばれる水平及び垂直のラインが付加されている。

【００４５】

通常はこの一連の計算ステップをアニメーションで見ることもできる。すなわち、何らかの処理が行われていることを示す点滅の移り変わり、すなわち計算ステップの流れが視覚的に表現される。また、アニメーションによる背景オブジェクトやコントロールストラクチャの動きは、計算の流れや内容をより分かりやすいものにする効果がある。これで計算スキームは定義されたので、あとはこの一連の計算ステップの各点滅しているノードで実行される具体的な処理の内容を定義すればよい。図１５の最初のフレームは初期化処理を表すもので、Ｌの対角線に１を代入し、Ｌの右上三角行列及びＵの左下三角行列に０を代入するという処理を行うが、この説明ではこれに対応した数式は省略する。

【００４６】

図１５のステップ２からステップ９までおよび図１６のステップ１０からステップ１７までに対応する具体的な計算式が図１７に示されている。図１７の上段の数式は、Ｕの現在点滅しているノードに対応する値を決定するものである。左辺はＵの現在点滅しているノードを表す。右辺の第一項はＡの現在点滅（参照を表す点滅）しているノードの値が参照されていることを示す。右辺の第２項では和の関数及び特別なステンシルが用いられているので、これを詳しく説明する。ここでは説明のために図１６のステップ１７に注目する。図１７の数式上段の右辺の大まかな概要は、（Ｌのある要素）×（Ｕのある要素）＋（Ｌのある要素）×（Ｕのある要素）＋（Ｌのある要素）×（Ｕのある要素）＋…であるが、このＬとＵの要素の掛け合わせる組み合わせがステンシルによって表されている。

【００４７】

図１６ステップ１７の文字を用いて説明すると、図１７上段の右辺、第二項の意味は、ａ・ｘ＋ｂ・ｙ＋ｃ・ｚである。このように掛け合わせる要素の組み合わせと順番がステンシル内の矢印によって示されている。このようにこの言語においては、計算スキームの各ステップにおけるノードの点滅のパタンや形状を利用して、視覚的で柔軟な計算式や関数の定義が可能である。

【００４８】

＜入出力処理」用の言語＞

「入出力処理」は、入出力処理用のマイクロ・アイコンとパネルによって表現される。マイクロ・アイコンとは、絵や文字を主体とした媒体でそれぞれ異なる情報を表す。パネルとは、マイクロ・アイコンを配置する盤であり、マイクロ・アイコンを組み合わせる役割をする。それらの例を図１８に示す。図１８の例では、ある入出力処理をマイクロ・アイコンとパネルによって定義している。パネルは３つのエリアから成り立ち、真ん中の大きい白いエリア、それより左側のエリア、右側のエリアとなる。

【００４９】

最初に真ん中のエリアについて説明する。このエリアには、実際に行われる入出力処理を定義するためのマイクロ・アイコンが配置される。左側の白い部分には処理全体の情報、真ん中の灰色（斜線ハッチング枠内）の部分にはソース（データを送る側）の情報、右側の灰色（斜線ハッチング枠内）の部分にはターゲット（データを受けとる側）の情報を表すマイクロ・アイコンが配置される。図１８の例では、処理全体の情報を表すマイクロ・アイコンが２つ、ソースの情報を表すマイクロ・アイコンが４つ、ターゲットの情報を表すマイクロ・アイコンが２つ、合計８つのマイクロ・アイコンから成り立ち、それぞれ次のような意味を表している（図１９）。

【００５０】

図１９で示すマイクロ・アイコンの説明とパネルにおけるマイクロ・アイコンの配置から総合的に判断すると、図１８の例で定義された入出力処理は次のようになる。図１９（ａ）と図１９（ｂ）から、１つのマスタープロセスから複数のスレーブプロセスへのスキャター・オペレーション、つまり、マスタープロセスからスレーブプロセスそれぞれに別のデータを送る処理であることが分かる。さらに、図１９（ｃ）と図１９（ｄ）から、マスタープロセスにはＳＢＵＦＦという名前の倍精度浮動小数点型の値を持つ三次元格子型構造体を持ち、それをＸ軸平面ごとに切り分け、左から順にスキャンし、データを送ることが分かる。

【００５１】

図１９（ｇ）から、その送られてきたデータを複数のスレーブプロセスが順にＲＢＵＦＦという名前の倍精度浮動小数点型の値を持つ二次元格子型構造体に格納していくことが分かる。また、図１９（ｅ）から、入出力処理を行う前にデータの型やサイズといった基本的な情報をやり取りすることで、ソース／ターゲット間に矛盾がないかをチェックすること、図１９（ｆ）から、入出力処理を行う前にプロセスの同期を計ること、図１９（ｈ）から、図１８の例で定義した入出力処理に関わるプロセスをＣＯＭＭという名前のコミュニケータが管理していることが分かる。このように、マイクロ・アイコンをパネルに配置することで具体的な意味を持ち、入出力処理を定義することができる。

【００５２】
次に左側のエリアについて説明する。このエリアには、真ん中の白いエリアで定義した入出力処理を定義しているノードに関するマイクロ・アイコンが配置される。図１８の例では、シンプルな四角形が１つ描かれたマイクロ・アイコンが配置され、１つのノードに定義された入出力処理であることを表している。
【００５３】
最後に右側のエリアについて説明する。このエリアには、ボタンが２つ配置されている。上部のボタンは、ＥＤＩＴ／ＨＥＬＰ（編集／ヘルプ）ボタンと言い、クリックするとＥＤＩＴモードとＨＥＬＰモードが入れ替わるようになっている。ＥＤＩＴモードのときは、上記で説明したマイクロ・アイコンをクリックすると、異なるマイクロ・アイコンに置き換えることができる。また、ＨＥＬＰモードのときは、マイクロ・アイコンをクリックすると、そのマイクロ・アイコンの説明を表示することができる（図２０）。
【００５４】
図２０に示すように、ヘルプは、マイクロ・アイコンが表す内容をアニメーションや音、テキストで表現し、ユーザの理解を助けている。下部のボタンは、Ｓｅｔ／Ｒｅｓｅｔボタンと言い、クリックするとＳｅｔモードとＲｅｓｅｔモードが入れ替わるようになっている。Ｓｅｔモードのときは、マイクロ・アイコンを編集することが可能で、ボタンを押すとそれらの設定を保存できる。そして、Ｒｅｓｅｔモードのときは、マイクロ・アイコンの編集は不可となり、ボタンを押すとＳｅｔモードに切り替わるようになっている。
【００５５】
図１８の例では、１つのマスタープロセスから複数のスレーブプロセスへの入出力処理を定義していた。その処理で送られてきたデータを元にスレーブプロセスがなんらかの計算を行うと想定すると、その結果をマスタープロセスへ返すための入出力処理があると考えられる。そこで、図１８の例に対応したスレーブプロセスからマスタープロセスへの入出力処理を定義する一例を図２１に示す。図１８の例と図２０の例の本質的な変更点を図２２に示す。この２つのマイクロ・アイコンの変更は、複数のスレーブプロセスから１つのマスタープロセスに結果を返す処理に変わったことをダイレクトに表現している。このように、マイクロ・アイコンを使用することで変更点を直感的に理解でき、読解にかかる時間を大幅に減らすことができる。
【００５６】
ここまで具体的な入出力処理の例をあげて説明してきたが、他にも様々なマイクロ・アイコンが存在する。一例を図２３に示す。図２３（ａ）のマイクロ・アイコンは、マスター／スレーブ間の処理を表している。図２３（ｂ）のマイクロ・アイコンは、同期処理に関する情報を表している。他にも、変数に関する情報を表すマイクロ・アイコンやデータの読み込み、格納に関する情報を表すマイクロ・アイコンなどが存在する。これら別の情報を表すマイクロ・アイコンを組み合わせることで、あらゆる入出力処理を表現することができる。
【００５７】
＜統合ビューと言語＞
計算手順を３つの要素、「計算のスキーム」、「計算式と変数」、「入出力処理」と特徴づけ、それぞれの要素をデータ・モデルに記述するための言語について説明した。計算手順をいくつかの要素に特徴づけることで、計算の複雑さを減少させ、コンポーネントの解析などを容易にする。このことからコンポーネントの変更や管理が容易に行えるという利点がある。また、これらの要素を統合することで、計算手順をよりコンパクトな形で表現することが可能となる。従って、自己説明型コンポーネントには、上記３つの要素を統合する統合ビューが存在する。後で説明するように、開発環境はこれらの要素をユーザの用途ごとに単独または統合して表示し、ユーザはそれらを操作することで自己説明型コンポーネントを変更・定義する。このとき、図２４に示すように、上記３つの要素と統合ビューは同期される。すなわち、統合ビューの変更は、対応する各要素に影響し、各要素の変更は統合ビューに影響する。

【００５８】
次に、統合ビューを記述するための言語について詳しく説明する。統合ビューは、特別なアイコンやテキストによって記述される。図２５に示すように、統合ビューは、処理の空間的関係を表わす構造体とそれに対応する変数が宣言されたヘッダが上部に配置される。構造体の宣言は、構造体を表わすアイコンとそれを特定するための属性から成る。ここでいう属性とは、例えば、構造体が二次元格子なら格子の縦・横のサイズ、ピラミッドならその深さなどのパラメータである。変数の宣言は、変数のタイプやデータタイプを特定するアイコン、変数名から成る。この変数が、対応する構造体に割り当てられる。図２６で示すように、計算上で用いる任意の数の構造体と変数の組をヘッダ内に宣言することが可能である。ヘッダの下に統合ビューの本体が配置される。この本体は、１つまたは複数の区分から構成されている。各区分の中で、対応する構造体における時間的関係を表わす処理の流れと式を定義する。また、区分は１つまたは複数の処理の流れを含み、上から下の順番で実行される。さらに処理の流れは、それを意味するアイコンと式から構成され、１つまたは複数の式を割り当てることができる。処理の流れをさらに部分的に分解するときは、式の変わりに区分を割り当てることができる。すなわち、本体は、計算アルゴリズムの構造に対応した階層的な構造を示し、それらは色（ここではハッチングパターンで示す）によって識別される。前述のように、ここでいう式とは、理論的・数学的な計算を特別な言語で記述したものである。

【００５９】
図２６はある計算アルゴリズムを表わす統合ビューの例である。画面の１番上に横に延びる長方形の領域がヘッダであり、主な構造体とそれに対応する変数が定義されている。この例では、いくつかの構造体と変数が宣言されているが、その中に二次元格子とそれに対応する変数Ｔが宣言されている。構造体は、二次元格子を表わすアイコンと二次元格子の属性、この場合はサイズを表わすＮ×Ｍというパラメータによって示されている。変数は、２つのアイコンと文字によって示されている。左のアイコンは、変数のタイプを表わす。この例では、二次元格子を表わしている。この他にも、一次元格子、木、ピラミッドなど様々なタイプが存在する。中央のアイコンは、変数のデータタイプを表わす。この例では、整数型を表わしている。その他にも、浮動小数点型、倍精度の整数・浮動小数点型、文字、文字列型などのデータタイプが存在する。そして、右側の文字Ｔが変数名である。

【００６０】
統合ビューの上から２段目以下は本体である。図示の統合ビューの例では、本体は、２段目（第一区分）と３段目（第二区分）２つの区分が構成されている。各区分は、左右２つの白い背景色をもった長方形の領域から成る。左側は、構造体とそれにおける処理の流れを表わすノードの点滅（実行）する順番を表示・定義する。右側は、その構造体の中のノードに割り当てる式、またはさらに階層的な区分を表示・定義する。

【００６１】
第一区分を具体的に説明すると、まず左側の領域は、左から順に、構造体を表わすアイコン、計算を実行する範囲を特定するためのマスクを表わすアイコン、処理の流れを示すアイコンである。この例における処理の流れは、本来は各行を上から下に処理し、また各行では左から右に処理をするというものだが、マスクのアイコンが追加されているので、一行目と一列目だけ計算を実行することを示している。このマスクの処理は省略できる。そして右側の領域が、具体的な式を表わしている。この例では、Ｔの点滅しているノードを０で初期化するということである。

【００６２】
第二区分を具体的に説明すると、左側の領域で定義している処理の流れは、第一区分とマスクのアイコンを除いて同じである。この場合、二次元格子の一行目と一列目を除いた部分だけ処理されることを意味する。右側の領域では、まず処理の流れを示すアイコンが配置され、二段階のｉｆ－ｅｌｓｅ構造であることが分かる。さらにその構造における具体的な式を、色と形で処理を区別し、それぞれに定義している。

【００６３】
［開発環境］
開発環境は、データ・モデルとマルチメディア言語群を用いて自己説明型コンポーネントへのアクセス手段を提供する。このとき、開発環境はユーザの用途ごとに的確なインターフェイスを構築し、作業を容易にする。例えば、あるユーザが計算アルゴリズムについて学ぶ際に、その基本的な構造体と処理の流れを理解するには、処理の空間的関係と時間的関係を表す要素、「計算のスキーム」を単独で表示することが非常に有効である。また、その処理における具体的な内容が知りたいときは、「計算のスキーム」という要素だけでは限界がある。このような場合、「計算式と変数」という要素と複合することで、目的を達成できる。このように、自己説明型コンポーネントの各特徴を捉えた要素を臨機応変に単独または複合したものをインターフェイスとして構築することで、ユーザの理解を助け、作業の効率化が図れる。このように、開発環境を用いることで、自己説明型コンポーネントを最大限に活かすことができ、より正確で深い理解へとつながる。
【００６４】
開発環境について、具体的な例をあげて説明する。ある最短経路を求めるためのアルゴリズムが自己説明型コンポーネントモデルによって、コンポーネント化されている。自己説明型コンポーネントは、いくつかの要素に特徴づけられているので、開発環境は、ユーザの用途に最も適したインターフェイスを即座に構築することができる。例えば、そのアルゴリズムにおける処理の流れを知りたい場合は、アルゴリズムに対応した構造体と処理の流れを表わす要素、「計算のスキーム」を単独で表示するインターフェイスを構築すればよい（図２７）。このとき、画像下の再生ボタンを押すことで、処理の流れをアニメーションで見ることができ、「計算のスキーム」が表わす情報を分かりやすく見ることができる。
【００６５】
また、あるアルゴリズムを自己説明型コンポーネントモデルによって、コンポーネント化するとき、開発環境は、ユーザが定義したい情報にあわせたインターフェイスを構築することができる。例えば、ユーザがある構造体とその構造体における処理の流れを定義したいとき、開発環境は、「計算のスキーム」を定義するのに最も適したインターフェイスを構築する（図２８）。図２８に示すように、各計算ステップを静的に捉え、画面下で一覧表示することで、処理の流れを明確に管理できるようにしている。そして、画面左上に現在編集中の計算ステップを表示し、その右隣にある編集用パネルで、ノード、イメージ、テキストなどを追加、削除し、定義している。
【００６６】
このように、同じ要素のインターフェイスでも、情報を見るとき、情報を定義するときでは、最適なインターフェイスは異なる。「計算のスキーム」を見るときでは、情報を動的に捉えることが肝要だが、「計算のスキーム」を定義するときでは、情報を静的に捉えることが重要である。このように、開発環境は、ユーザの用途に合わせ、様々なインターフェイスを構築することができる。
【００６７】
次に、本発明に係るコンピュータ処理方法、コンピュータ処理システムの一実施の形態を説明する。この実施の形態では、コンポーネントに関連する情報を複数の項目ごとに分けて保存できるコンポーネント構造を採用する。
【００６８】
このコンポーネント構造はアルゴリズムサイバーフィルムと呼ばれるものであり、一連のマルチメディアフレームからなる物理構造を持つマルチメディアデータ形式である。アルゴリズムサイバーフィルム（以下、フィルムと呼ぶ）は、ソフトウェアのアルゴリズム的性質に主眼を置くコンポーネント構造である。例えば、ある物体の一つの性質を一枚のフレームで表し、フィルム全体ではそのオブジェクトのたくさんの性質を表すことを可能とする構造である。必要に応じて、意味的に関連するフレームをマルチメディアシーンという単位でくくることができる。

【００６９】
各マルチメディアフレームにはマルチメディアオブジェクトである、図形、画像、動画、音、文字が配置され、これらオブジェクトの空間的・時間的な状態によってマルチメディアフレームが定義されるという表現手法が用いられている。
【００７０】
図２９は、計算アルゴリズムを表現する一連のシーン、フレームの例を示す。図２９の上段は第１シーン（シーン１）の第１～第５フレーム（フレーム１～５）を左から順に示している。また、図２９の下段は第２シーンの第１～第５フレームを左から順に示している。なお、この図２９の上段は図７と共通である。各フレームは前記マルチメディアオブジェクトとして、二次元格子状のノードの集合を含んでいる。この例では、一連のフレームは二次元格子上の計算の流れを表し、各フレームはその１ステップを表現している。各フレーム内で、どのような処理がどのような順序で実行されるかを定義する表現手法は以下のようになっている。
【００７１】
まず、ノードの色がそのノードで実行される処理を表す。次に、各ノードが実行される順序はそのノードの点滅によって表す。さらに、点滅の種類で実行される処理のおおまかな性質を表す。点滅の種類は例えば、前出の図５に示すようなものがある。
【００７２】
さらに、処理が済んだノードをマークし、処理の「流れ」を読み取りやすくするために背景色を用いる。ただし図２９では背景色をハッチングで示している。図２９の例で、第１シーンでは、二次元格子のデータ構造において、各列において左から右に順に何らかの計算をし、各計算は一つ前の列（左側の列）の値を参照し現在の列の処理を実行している。また、第２シーンでは、逆に右から左に順に計算を行ない、各計算は一つ前の列（右側の列）の値を参照し現在の列の処理を実行している。
【００７３】
また、各オブジェクトが表す意味の抽象度を下げ、ユーザの理解を助けるために、図３０（ｂ）または図２９に示すようにバックグラウンドオブジェクトやフォアグラウンドオブジェクトを付け加えることができる。図３０（ａ）のフレームはバックグラウンドオブジェクトやフォアグラウンドオブジェクトを使用する前の状態を示し、図３０（ｂ）のフレームはバックグラウンドオブジェクトやフォアグラウンドオブジェクトを使用した状態を示している。
【００７４】
同様にして、音を用いて処理の流れや内容・特徴を表すこともできる。例えば、シーンの変わり目で効果音（例えば、チャイムの音）を鳴らしたり、点滅の種類（実行される処理の種類）に応じて異なった音を鳴らすことができる。
【００７５】
これまでに、計算アルゴリズムを実装するソフトウェアコンポーネントについて、その意味特徴をコンポーネントの概要、内部処理の流れ、処理内容の詳細な定義、入出力様式、コンポーネント同士の関連情報、作者著作権情報の各観点に分けてマルチメディアフレームを分類する方法と、コンポーネントの概要と内部処理の流れを定義するための表現手法が提案されてきた。
【００７６】
本実施の形態では、本コンポーネント構造を、あらゆるソフトウェアコンポーネントを表現できるように拡張するとともに、本コンポーネント構造を持ったソフトウェアコンポーネントを検索、閲覧、生成、編集、実行するための新たなコンポーネント管理システムの構造を備える。
【００７７】
コンポーネント構造の拡張として、上に挙げた六つの観点はコンポーネントを構成する基本情報として記録する一方、必要に応じて（コンポーネントの種類やアプリケーションに応じて）新たな項目を定義し、追加することを可能とする。

【００７８】
コンポーネント管理システムは、上記フィルムに対して検索、閲覧、生成、編集、実行などの操作を行なうシステムである。本システムは、前記コンポーネントについて、そのマルチメディアシーンおよびマルチメディアフレームを読み込み、予め定義され情報項目ごとに専用の表現手法を用いてそれぞれのマルチメディアフレームを描画する。
【００７９】
システム上でのユーザ操作の種類や目的に応じ、描画済みのマルチメディアフレームから適切なものを取り出し、必要に応じて二つ以上のマルチメディアフレームを組み合わせて、画面に表示することができる。
【００８０】
図３１は、マルチメディアフレームの組み合わせからビューを構成する仕組みを示している。図３１（ａ）に示すように、情報項目ごとに一連のフレームを含んだシーンが配置されている。これらの中から複数のマルチメディアフレームを選んで組み合わせて図３１（ｂ）、（ｃ）、（ｄ）に示すようなビューを構成する。さらに、各情報項目は特有のビューを持っている。図３２（ａ）は、内部処理の流れを再生するためのビューで、図３２（ｂ）は内部処理の流れを並べて表示するためのビューである。図３２（ａ）に示すように、これらのビューは必要に応じてさらに複数のビューに分割して表示することが可能である。さらに図３３は、内部処理の流れの内容を変更するためのビューである。
【００８１】
上記機能により、コンポーネントが持つ複雑で多量の情報を一ヵ所に保ったまま、用途や好みに応じて、取捨選択しやすくすることでそのコンポーネントに対する理解を容易にし、ひいてはそのコンポーネントの再利用を強力に促すという特質を持つ。
【００８２】
このように、複雑に関係し合う情報を、アプリケーションに応じた適切な分類に従って整理し、必要に応じて提供できるようにすることで、状況に応じて最も効果的なユーザインタフェースを構築することを可能とする。
【００８３】
それぞれの情報項目ごとに最適なマルチメディア表現を利用することができる。このマルチメディア表現は、ユーザの理解を助ける機能と情報を定義する機能の両方を持つものであり、その意味においてマルチメディア言語と考えることができる。この言語は、フィルムのマルチメディアフレームとマルチメディアオブジェクトによって表現できる空間的、時間的概念を基礎とし、オブジェクトの空間的・時間的な状態によって意味が定義される。
【００８４】
このような言語を情報項目ごとに定義し、切り替えて使えるようにすることで、幅広い意味を定義するのに必要な抽象性と、容易な理解に必要な具体性を適度なバランスで持ち合わせる一連の言語によってコンポーネントを表現することを可能としている。これらの言語は、基本的にはマルチメディアなものであり、これまでの文字や挿絵を中心にした表現手法に比較して、実態に近い表現が可能であり、表現の過程で生まれる抽象化の度合いを低く抑える効果がある。
【００８５】
各情報項目のマルチメディアフレームはそのフレーム番号によって同期される。すなわち、複数の情報項目を一つの画面に表示する際、同一のフレーム番号のマルチメディアフレームをもってくることで、その時間におけるコンポーネントの状態を複数の観点から見ることが可能となる。
【００８６】
また、本コンポーネント管理システムは、一つの情報項目に関して加えられた編集操作を受け、予め定められた変換規則に従って、関連する情報項目のマルチメディアフレームに必要な編集を自動的に加える同期機構を含んでいる。
【００８７】

サイバーフィルムとして生成されるコンポーネントは通信ネットワークを介してアクセス可能なデータベースに格納され、本データベースは上記コンポーネント管理システムの一部である。各コンポーネント管理システムは一つ以上のデータベースにアクセスし、コンポーネントの検索と保存を行なう。

【００８８】

図３４は、本発明が実現する自己説明型コンポーネント開発の流れを示したものである。自己説明型コンポーネント開発は、共有コンポーネントライブラリが中心となっており、その基礎は、特別の認定を受けた団体・開発者による基礎コンポーネント開発によって作成され、認定を受けたものである。この認定では、コンポーネントとしての妥当性や、ある一定の水準の自己説明性に達しているかのチェックなどを行なう。

【００８９】

自己説明型コンポーネント開発は、このコンポーネントライブラリから一つまたは複数のコンポーネントを組み合わせたものを基礎に開発が行なわれる。完成したコンポーネントは、認定を受けたのち、新たにコンポーネントライブラリに登録し、共有することができる。個人的に再利用するために保存し、共有ライブラリに納めないものは認定を受ける必要がない。内部に認定を受けていないコンポーネントを一つでも含むコンポーネントは共有ライブラリに含むことができない。

【００９０】

次に、本発明の実施の形態をさらに具体的に説明する。図３５は、本発明のハードウェアシステム構成の一実施の形態を示すものであるが、本発明では、特別のハードウェア構成に限定されるものではなく、現在一般的に使用されているネットアクセスが可能なパーソナルコンピュータ（またはワークステーション）１０を用いる。パーソナルコンピュータ１０は、コンピュータ本体１とユーザインタフェース２１などからなる。ユーザインタフェース２１は、ディスプレイ２と、キーボード３と、マウス４などからなる。パーソナルコンピュータ１０は、通信ネットワーク１２を介してデータベース５にアクセスが可能である。

【００９１】

この構成において、本実施の形態は、サイバーフィルムによるアルゴリズムの定義および学習のためのシステムを機能させるプログラムをコンピュータ本体１に導入して形成される。

【００９２】

本発明に係るコンポーネント管理システムの一実施の形態として、システムの機能構成図を図３６に示す。これは、数値計算用アプリケーション・ソフトウェアの構築を対象としたシステムの場合についての説明である。システムは、ユーザインタフェース２１、各モジュールを制御するコントローラ２２、コンポーネントを保存するデータベース５、サイバーフィルムのマルチメディアフレームの構築、取得、編集等を行なうフレーム管理モジュールＭ１～Ｍｎ、マルチメディアフレームを描画するレンダリングエンジン２４から形成される。

【００９３】

モジュールＭ１は内部処理の流れ（アルゴリズムスケルトン）を、モジュールＭ２は処理内容の詳細な定義（変数や式）を、モジュールＭ３は入出力様式を、モジュールＭ４はコンポーネントの概要を、モジュールＭ５はコンポーネントの関連情報を、モジュールＭ６は作者著作権情報に関するマルチメディアフレームを、それぞれ管理するモジュールである。仮に、あるアプリケーションが追加の情報項目を必要とする場合、対応するｘ番目のモジュールＭｘを追加する。

【００９４】

本発明に係るコンポーネント管理システムの一実施の形態の制御手順を図３７に示す。システムを起動（Ｓ１）するとメイン画面が表示される（Ａ１）。ユーザからサイバーフィルムを開く指示がある場合（Ｂ１）はモジュールＭ２によりデータベース等からサイバーフィルムを検索し読み込み（Ａ２）、サイバーフィルムからマルチメディアフレームが

展開され、それぞれ適切な管理モジュールＭ１～Ｍｎにロードされる。これでユーザが選択したコンポーネントを見たり、編集したりする準備が完了する。

【００９５】

次に、インターフェイス２１でユーザの操作により、アルゴリズムスケルトンを見るという指示（ＢＷ１）があれば、インターフェイス２１に内部処理の流れ（アルゴリズムスケルトン）表示画面を表示する（ＡＷ１）。変数／式を見るという指示（ＢＷ２）があれば、インターフェイス２１に変数／式の表示画面を表示する（ＡＷ２）。入出力を見るという指示（ＢＷ３）があれば、インターフェイス２１に入出力の表示画面を表示する（ＡＷ３）。

【００９６】

この例では、おおまかに３種類の表示モード（ＢＷ１～ＢＷ３）があり、それぞれに一つずつの表示画面（ＡＷ１～ＡＷ３）を含んでいるが、表示モードの種類とそれらに含まれる表示画面の数に制限はない。また、各表示画面は、一つ以上のフレーム管理モジュール（Ｍ１～Ｍ６）が保管している一つ以上のフレームをレンダリングエンジン２４が予め定められた規則に従って描画したものである。

【００９７】

同様にして、ユーザの操作によりアルゴリズムスケルトンを変更するという指示があれば（ＢＥ１）、アルゴリズムスケルトンの変更画面を表示する（ＡＥ１）。また、ユーザのマウスおよびキーボードからの入力によって現在のサイバーフィルムのアルゴリズムスケルトンを変更・更新し、変数／式を変更するという指示があれば（ＢＥ２）、変数／式の変更画面を表示する（ＡＥ２）。また、ユーザのマウスおよびキーボードからの入力によって現在のサイバーフィルムオブジェクトの変数／式を変更・更新し、入出力を変更するという指示があれば（ＢＥ３）、入出力の変更画面を表示する（ＡＥ３）。

【００９８】

これらの表示の後に、ユーザのマウスおよびキーボードからの入力によって、現在のサイバーフィルムオブジェクトの入出力を変更・更新する。これら編集操作は、コントローラ２を通して各フレーム管理モジュール（Ｍ１～Ｍ６）に伝達される。一つの編集操作が異なる情報項目に属する複数のマルチメディアフレームに影響を与える場合は、コントローラ２が該当するフレーム管理モジュール（Ｍ１～Ｍ６）に必要な編集命令を伝達する。

【００９９】

次に、現在のサイバーフィルムを保存するという指示があれば（Ｂ４）、コントローラ２２により現在のサイバーフィルムをデータ変換しデータベースまたはファイルに保存する（Ａ４）。終了の指示がある場合（Ｂ５）はシステムを終了し（Ｓ２）、終了しない場合は分岐点Ｂ１に戻る。

【０１００】

本実施の形態によれば、コンポーネントを理解し利用するのに必要な情報をコンポーネント内に記述できるコンポーネント構造が採用され、それから提供される情報から適切なものを適時取り出し表示することができ、それぞれの情報項目ごとに最適なマルチメディア表現を利用できる。これにより、コンポーネントの利用（理解および作成）という観点から、各コンポーネントの自己説明性が向上する。さらに、結果として開発されたソフトウェア・システムもまた高い自己説明性を持つ。

【図面の簡単な説明】

【０１０１】

【図１】本発明に係る自己説明型コンポーネントモデルの３つの柱を示す模式図。

【図２】本発明に係るコンポーネントモデルのデータ・モデルにおける空間的関係と時間的関係の例を示す説明図。

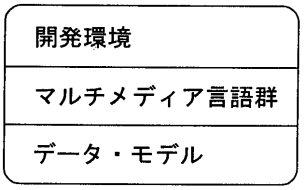
【図３】本発明に係るコンポーネントモデルのマルチメディア言語における構造体の例を示す説明図。

【図４】本発明に係るコンポーネントモデルのマルチメディア言語において処理を色で区別する例を示す説明図。

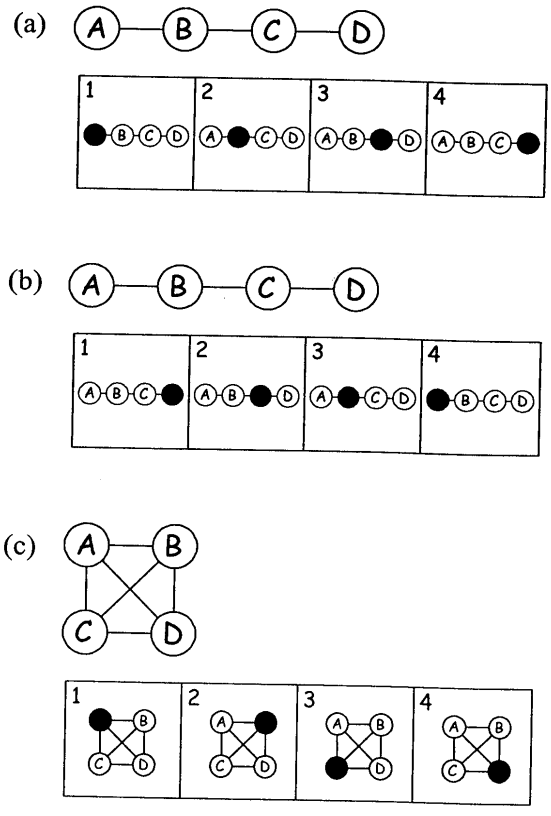
【図５】本発明に係るコンポーネントモデルのマルチメディア言語において点滅の種類の例を示す説明図。
【図６】本発明に係るコンポーネントモデルのマルチメディア言語において点滅の種類の例を示す説明図。
【図７】本発明に係るコンポーネントモデルのマルチメディア言語における計算スキームの例を示す説明図。
【図８】本発明に係るコンポーネントモデルのマルチメディア言語におけるグループオブジェクトの使用例を示す説明図。
【図９】本発明に係るコンポーネントモデルのマルチメディア言語における前景オブジェクトの使用例を示す説明図。
【図１０】本発明に係るコンポーネントモデルのマルチメディア言語における背景オブジェクトの使用例を示す説明図。
【図１１】本発明に係るコンポーネントモデルのマルチメディア言語における変数の参照の例を示す説明図。
【図１２】本発明に係るコンポーネントモデルのマルチメディア言語におけるステンシルの例を示す説明図。
【図１３】本発明に係るコンポーネントモデルのマルチメディア言語におけるシステム関数の例を示す説明図。
【図１４】本発明に係るコンポーネントモデルのマルチメディア言語におけるＣ言語によるＬＵ分解の実装の例を示す説明図。
【図１５】本発明に係るコンポーネントモデルのマルチメディア言語におけるＬＵ分解の計算スキームの例を示す説明図であって、ステップ１からステップ９までを示す図。
【図１６】本発明に係るコンポーネントモデルのマルチメディア言語におけるＬＵ分解の計算スキームの例を示す説明図であって、図１５に続くステップ１０からステップ１７までを示す図。
【図１７】本発明に係るコンポーネントモデルのマルチメディア言語におけるＬＵ分解の数式の例を示す説明図。
【図１８】本発明に係るコンポーネントモデルのマルチメディア言語における入出力処理をマイクロ・アイコンとパネルで定義する例を示す説明図。
【図１９】図１８のマイクロ・アイコンの意味を示す説明図。
【図２０】本発明に係るコンポーネントモデルのマルチメディア言語におけるヘルプの例を示す説明図。
【図２１】図１８の例に対応した入出力処理の例を示す図。
【図２２】図１８の例と図２１の例の比較を示す説明図。
【図２３】本発明に係るコンポーネントモデルのマルチメディア言語におけるマイクロ・アイコンの例を示す説明図。
【図２４】本発明に係るコンポーネントモデルの計算手順の要素と統合ビューの関係を示す説明図。
【図２５】本発明に係るコンポーネントモデルの統合ビューの配置を示す説明図。
【図２６】本発明に係るコンポーネントモデルの統合ビューの例を示す図。
【図２７】本発明に係るコンポーネントモデルの計算スキーム用インターフェイスの例を示す図。
【図２８】本発明に係るコンポーネントモデルの計算スキーム定義用インターフェイスの例を示す図。
【図２９】本発明に係るコンピュータ処理方法における計算アルゴリズムを表現する一連のシーンおよびフレームの例。
【図３０】本発明に係るコンピュータ処理方法における計算アルゴリズムを表現するフレームにバックグラウンドオブジェクトやフォアグラウンドオブジェクトを付加する例を示す図で、（ａ）はこれらオブジェクトを付加する前の表示の例を示し、（ｂ）はこれらオブジェクトを付加した後の表示の例を示す。

【図３１】本発明に係る自己説明型コンポーネントモデルによるコンピュータ処理方法におけるマルチメディアフレームの組み合わせからビューを構成する例を示す図で、（ａ）は情報項目ごとに一連のフレームを含んだシーンが配置された状態を示す模式的斜視図であり、（ｂ）、（ｃ）、（ｄ）はそれぞれ、（ａ）の複数のフレームからマルチメディアフレームを選んで組み合わせて表示するビューを示す。
【図３２】本発明に係る自己説明型コンポーネントモデルによるコンピュータ処理方法における内部処理の流れを再生するためのビューの例を示す図であって、（ａ）は内部処理の流れを再生するためのビュー、（ｂ）は内部処理の流れを並べて表示するためのビュー。
【図３３】本発明に係る自己説明型コンポーネントモデルによるコンピュータ処理方法における内部処理の流れの内容を変更するためのビュー。
【図３４】本発明に係る自己説明型コンポーネントの開発の流れを示す説明図。
【図３５】本発明に係る自己説明型コンポーネントモデルによるコンピュータ処理システムの一実施の形態のハードウェア構成を示すブロック図。
【図３６】本発明に係る自己説明型コンポーネントモデルによるコンピュータ処理システムの一実施の形態の機能構成を示すブロック図。
【図３７】本発明に係る自己説明型コンポーネントモデルによるコンピュータ処理方法の一実施の形態の制御手順を示すフローチャート。
【符号の説明】
【０１０２】
１…コンピュータ本体、２…ディスプレイ、３…キーボード、４…マウス、５…データベース、１０…パーソナルコンピュータ、１２…通信ネットワーク、２１…ユーザインタフェース、２２…コントローラ、２４…レンダリングエンジン、Ｍ１～Ｍｎ…管理モジュール。

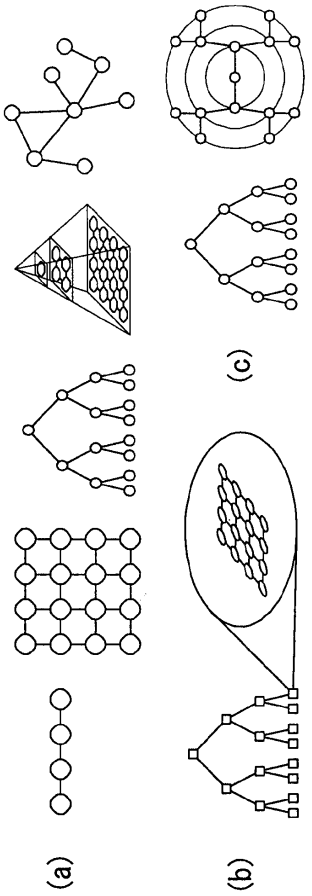
【図１】



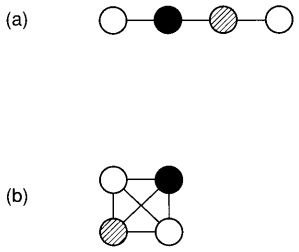
【図２】



【図３】


(a)
(b)
(c)

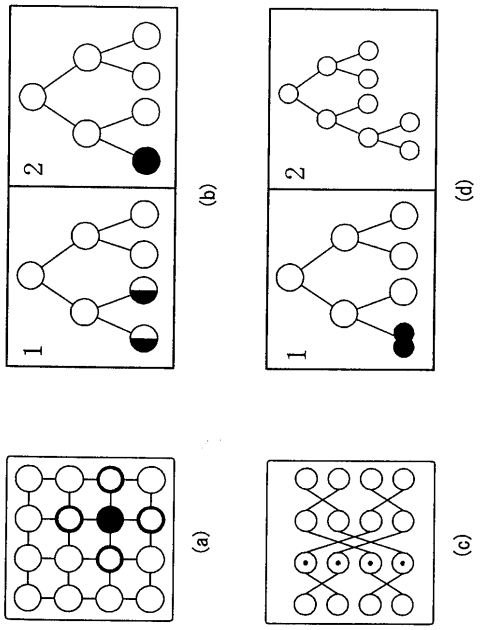
【図４】


(a)
(b)

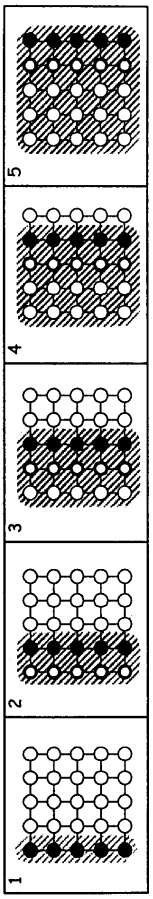
【図５】


(a)
◆実行を表す。
ノードに定義された処理を実行していることを表す。
(b)
◆参照を表す。
一般的には、他のノードの処理の中でこのノードが利用されている状態を示す。
(c)
◆次のステップで実行するノードを選択するための処理を表す。
条件によっては次のステップで実行するノードが違う場合に用いる。
(d)
◆前のステップの処理がまだ実行中であることを表す。
基本的には、各処理はそのステップ内で終了する同期処理であるが非同期処理を表すときにこの点線を用いる。
(e)
◆他処理の結果を待っている状態を表す。
これもステップによる同期ではなくデータや他の処理の結果に同期するために用いる。
(f)
◆構造体を変形する処理を表す。
処理の結果に基づいて構造体の構成を動的に変更する時に用いる。

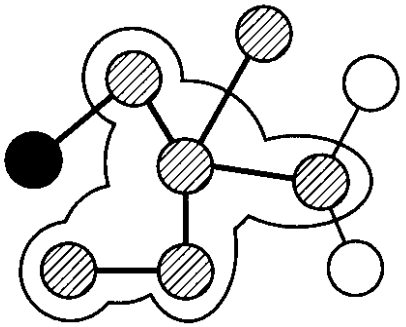
【図６】


1
2
(a)
(b)
(c)
(d)

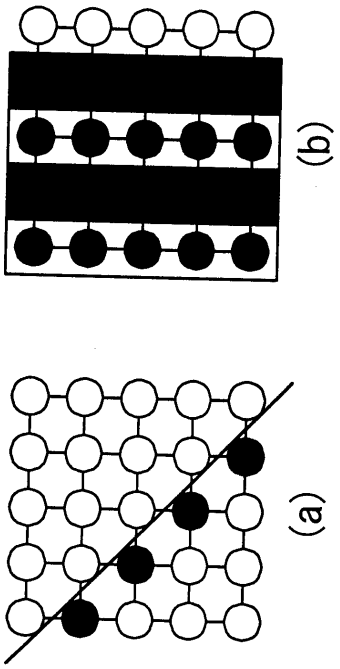
【図７】



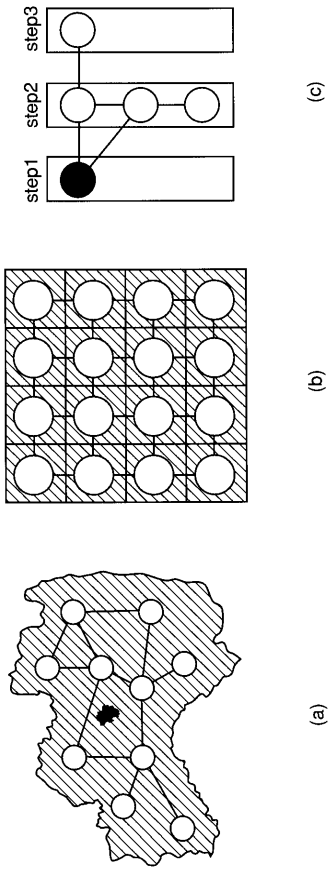
【図８】



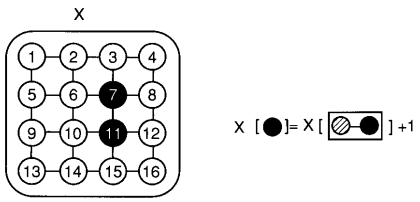
【図９】



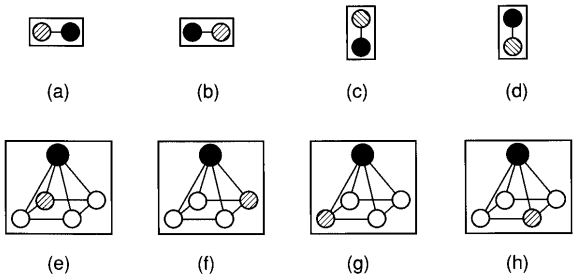
【図１０】



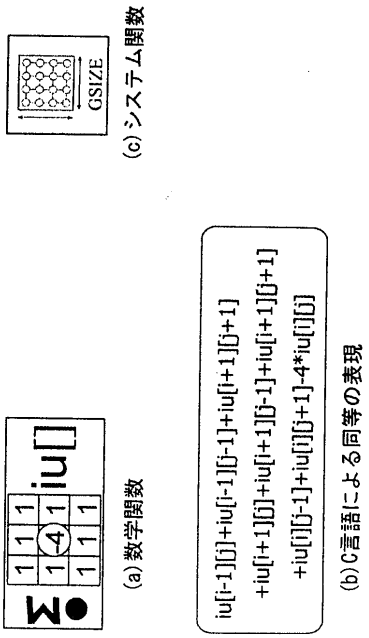
【図１１】



【図１２】



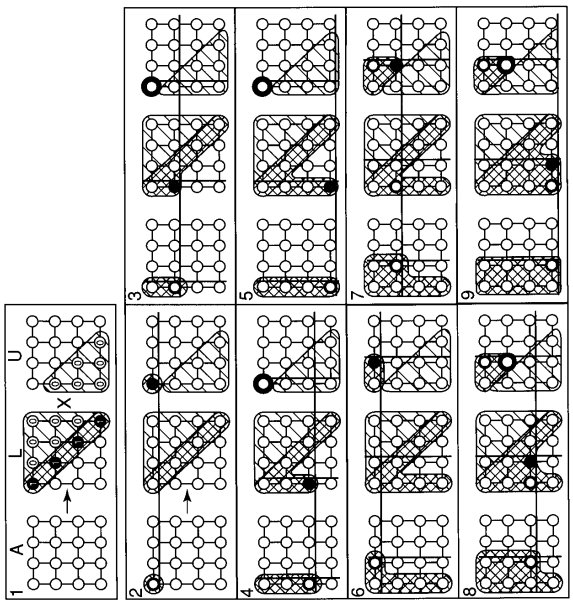
【図１３】



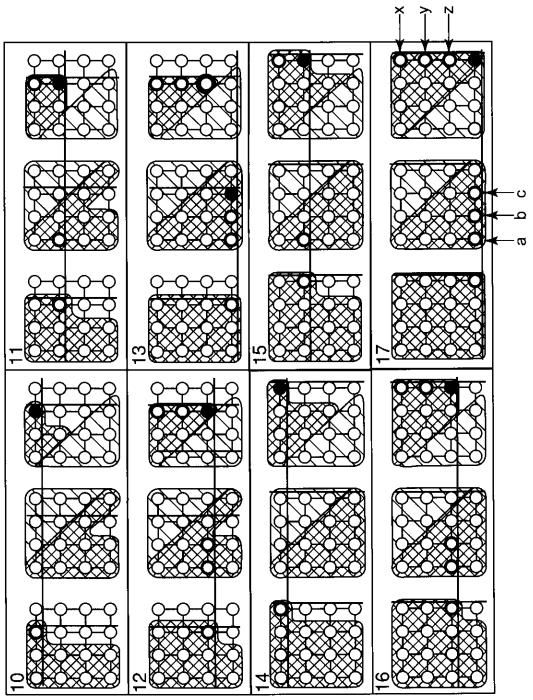
【図１４】

```
void LUDecomposite(){
        for ( int i = 0; i < N; i++ ){
                for ( int j = 0; j < N; j++ ){
                        L[i][j] = U[i][j] = 0;
                        if ( i == j ) L[i][j] = 1;
                }
        }

        for ( int j = 0; j < N; j++ ){

                for ( int i = 0; i <= j; i++ ){
                        double sum = 0.0;
                        for ( int k = 0; k <= i - 1; k++ ){
                                sum += L[i][k] * U[k][j];
                        }
                        U[i][j] = A[i][j] - sum;
                }

                for ( int i = j + 1; i < N; i++ ){
                        double sum = 0.0;
                        for ( int k = 0; k <= j - 1; k++ ){
                                sum += L[i][k] * U[k][j];
                        }
                        L[i][j] = ( A[i][j] - sum ) / U[j][j];
                }
        }
}
```

【図１５】



【図１６】

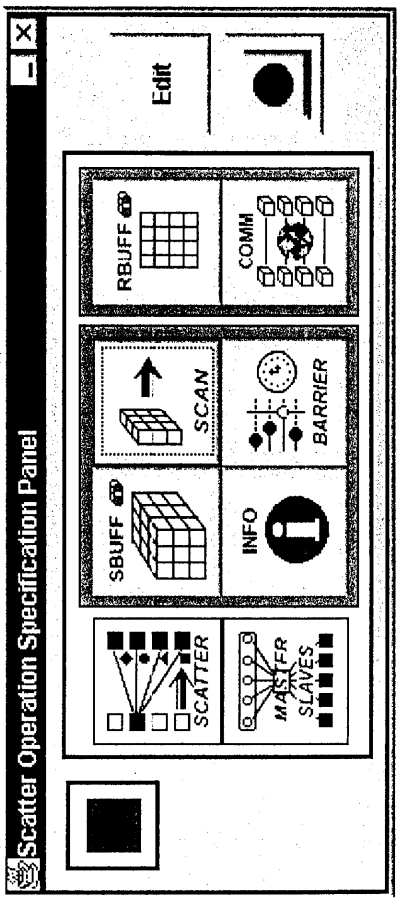


【図１７】

$$U[\bullet] = A[O] - \Sigma(L[\square] \times U[\square])$$

$$L[\bullet] = \frac{1}{U[O]}\left(A[O] - \Sigma(L[\square] \times U[\square])\right)$$

【図１８】



【図１９】

(a) 

１つのプロセスから複数のプロセスへのスキャター・オペレーションであることを表す。

(b) 

マスター／スレーブの関係にあるプロセス間の処理であることを表す。

(c) 

SBUFFという名前の倍精度浮動小数点型の値を持つ3次元格子の構造体であることを表す。

(d) 

3次元格子の構造体を、X軸平面ごとに分割して左から順にスキャンするということを表す。

(e) 

データの型やサイズといった入出力処理における基本的な情報を設定したことを表す。

(f) 

入出力処理を行う前に同期を計っていることを表す。

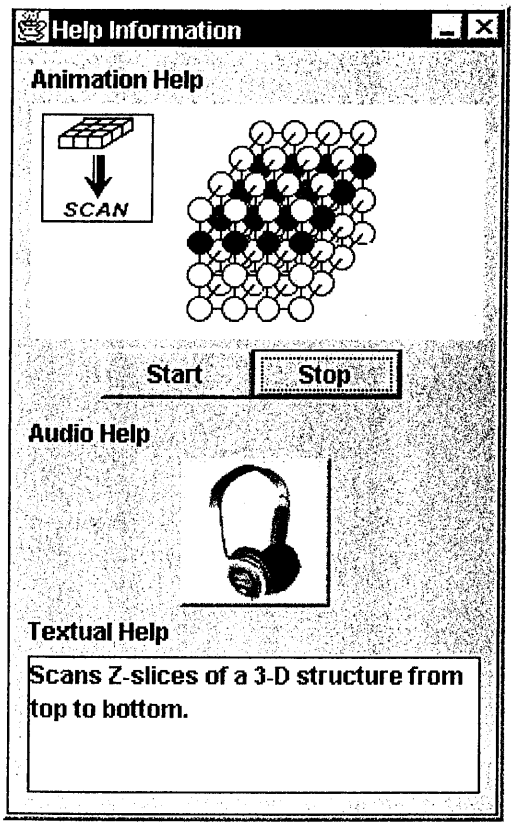
(g) 

RBUFFという名前の倍精度浮動小数点型の値を持つ2次元格子の構造体であることを表す。

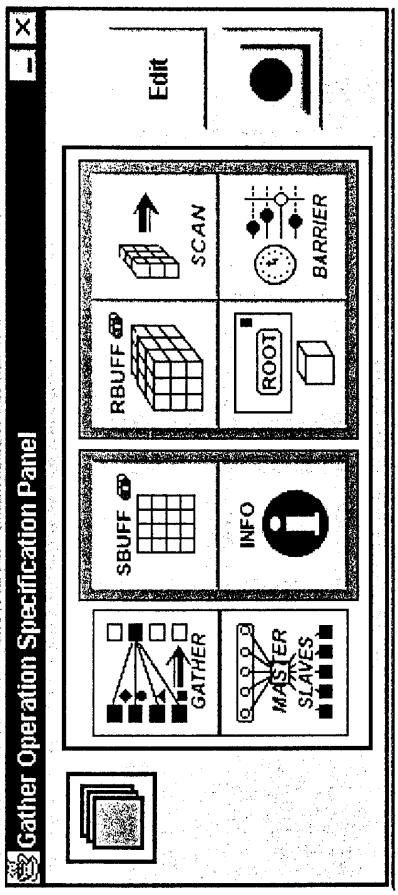
(h) 

入出力処理で実行されるプロセスを識別するためにCOMMという名前のコミュニケータを設定したことを表す。

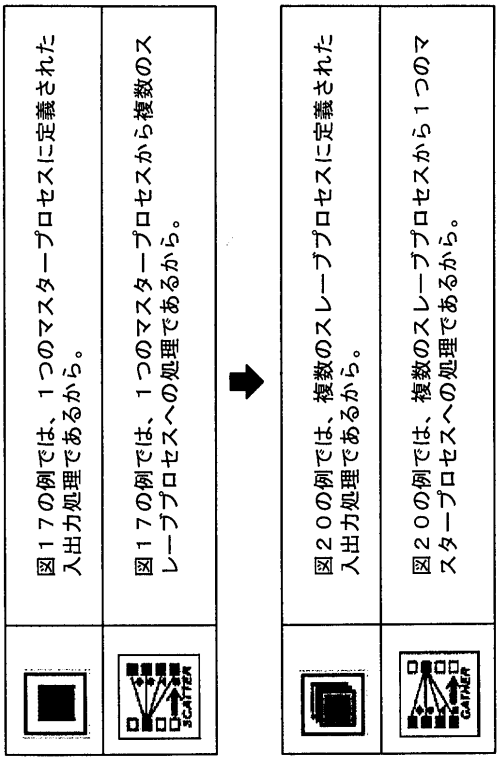
【図２０】



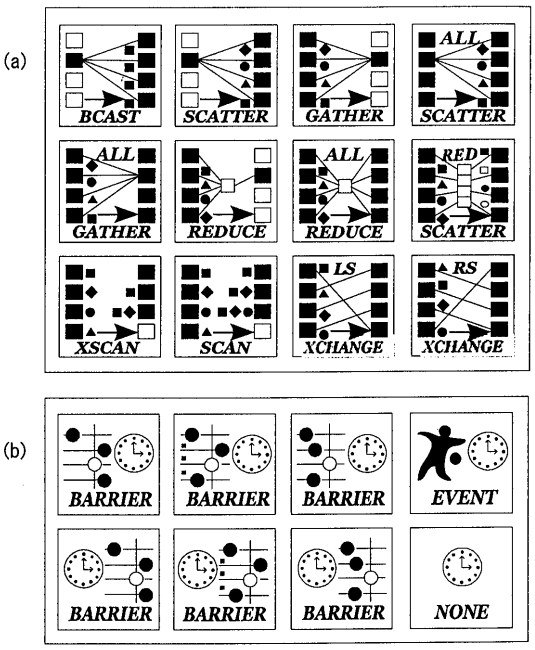
【図２１】



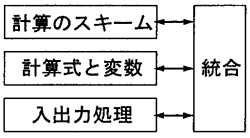
【図２２】



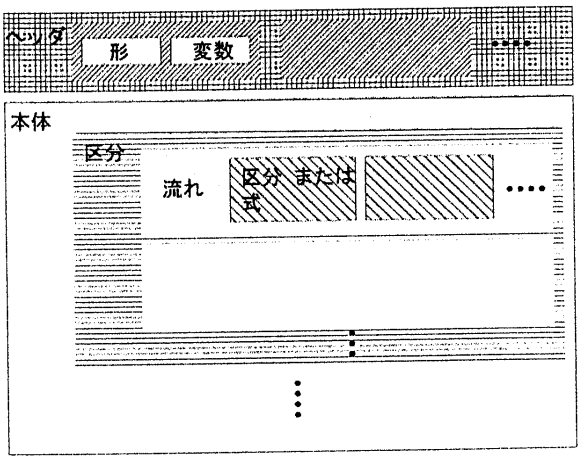
【図２３】



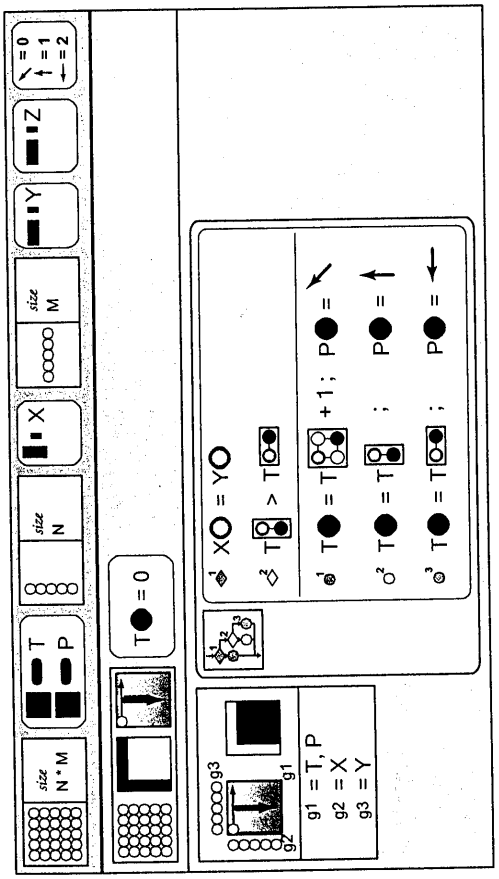
【図２４】



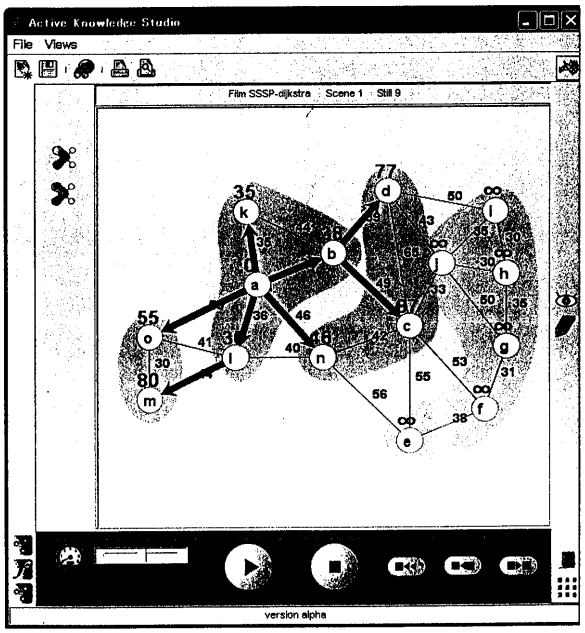
【図２５】



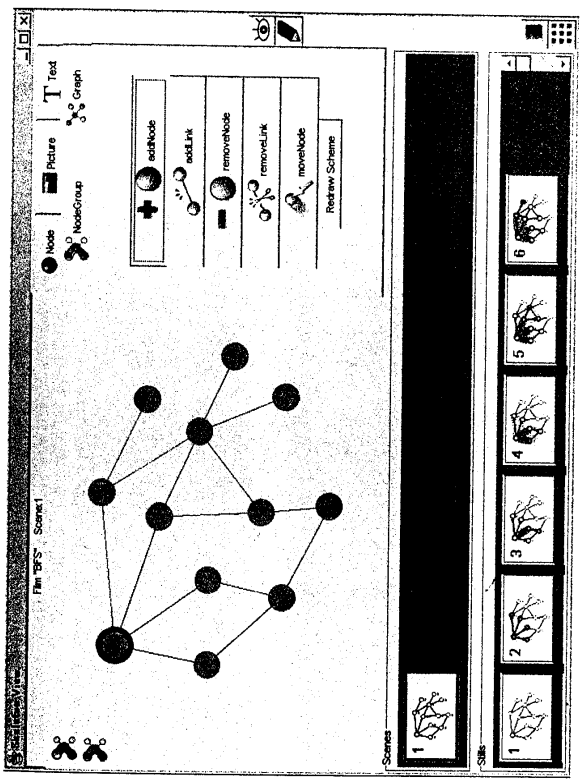
【図２６】



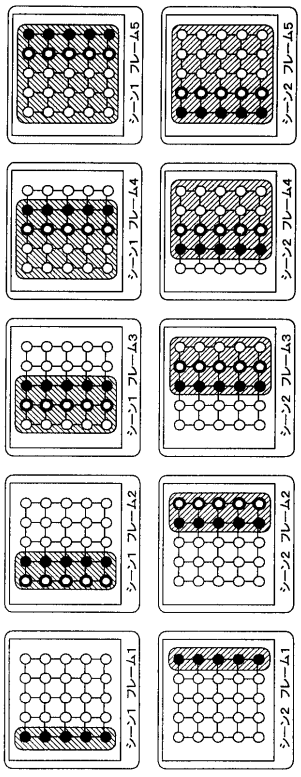
【図２７】



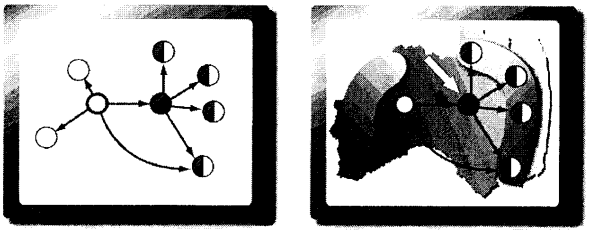
【図２８】



【図２９】

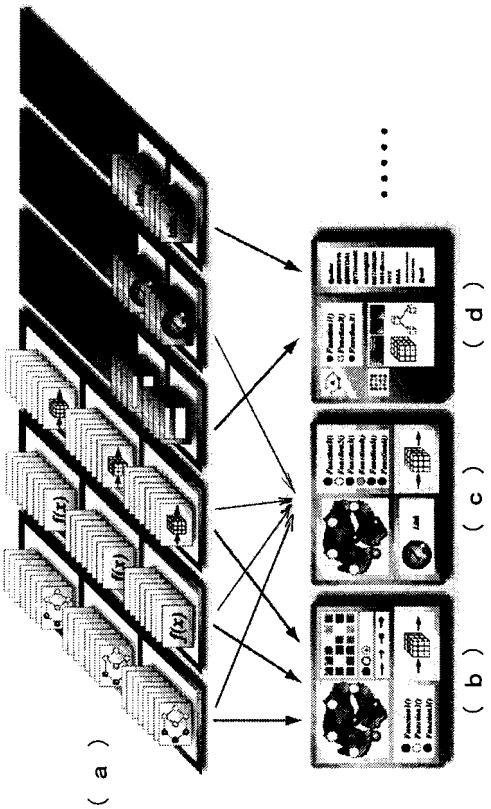


【図３０】

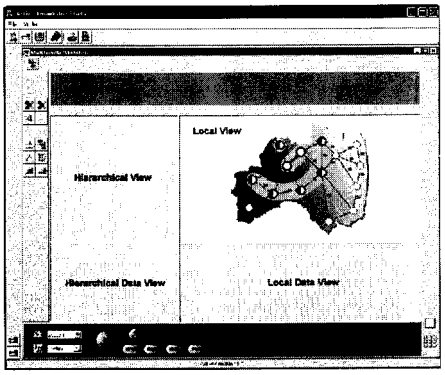


（ａ）　（ｂ）

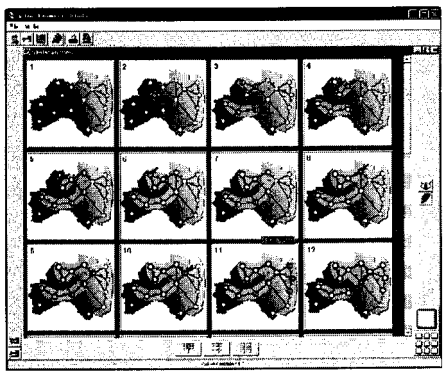
【図３１】



【図３２】

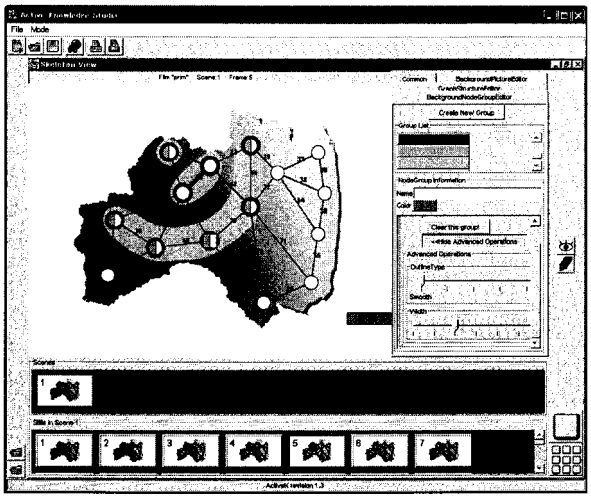


（ａ）

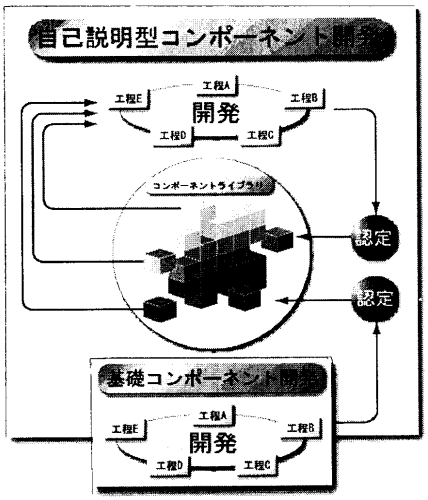


（ｂ）

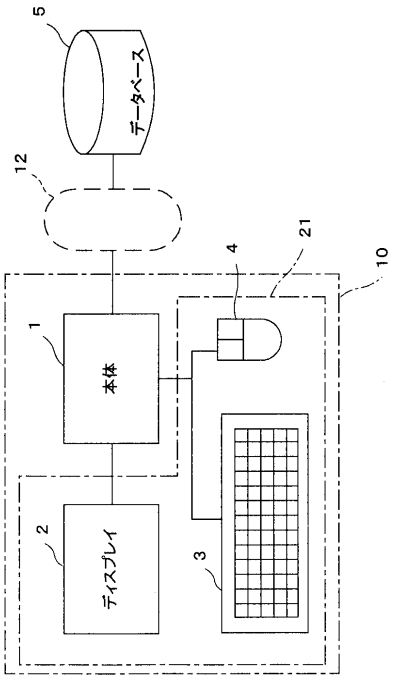
【図３３】



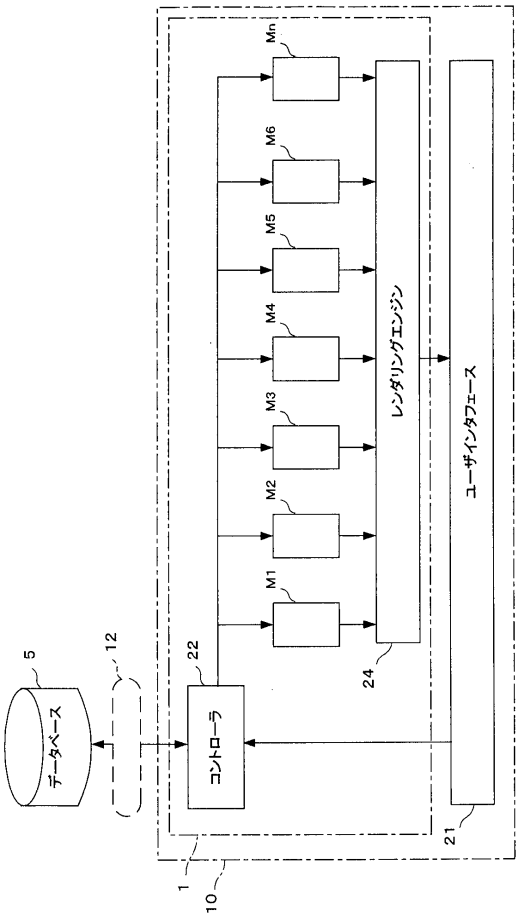
【図３４】



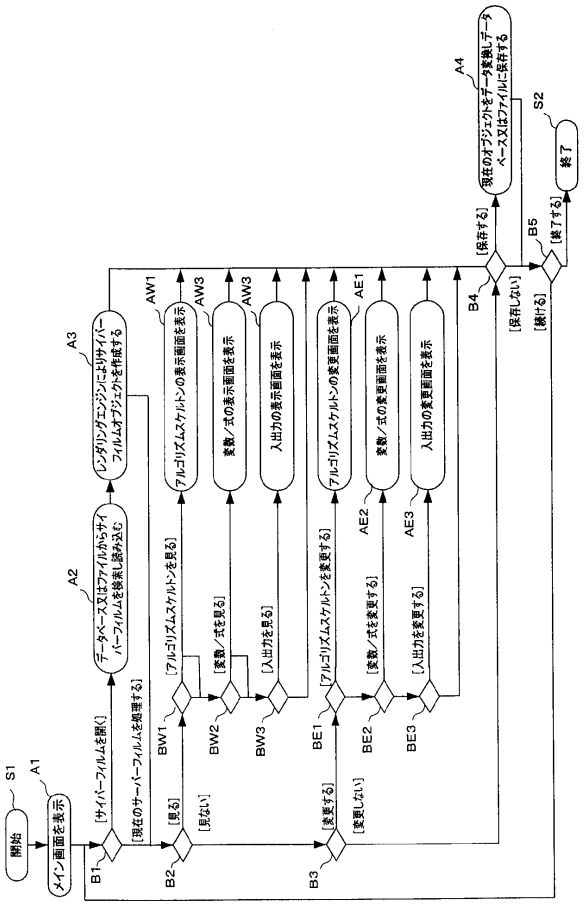
【図３５】



【図３６】



【図３７】



フロントページの続き

審査官　漆原　孝治

(56)参考文献　特開平０８－１２３６７７（ＪＰ，Ａ）
特開２００２－３０４４２８（ＪＰ，Ａ）
特開２００３－０８４８７６（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　　９／４４