(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号

特許第5269450号
(P5269450)

(45)発行日　平成25年8月21日(2013.8.21)　(24)登録日　平成25年5月17日(2013.5.17)

(51)Int.Cl.　　F I
G06F 11/22 (2006.01)　　G06F 11/22 310A
G01R 31/28 (2006.01)　　G01R 31/28 H

請求項の数 8　(全 17 頁)

(21)出願番号　特願2008-72401(P2008-72401)
(22)出願日　平成20年3月19日(2008.3.19)
(65)公開番号　特開2009-230272(P2009-230272A)
(43)公開日　平成21年10月8日(2009.10.8)
審査請求日　平成23年2月9日(2011.2.9)

(73)特許権者　390005175
株式会社アドバンテスト
東京都練馬区旭町１丁目３２番１号
(74)代理人　100079108
弁理士　稲葉　良幸
(74)代理人　100093861
弁理士　大賀　眞司
(74)代理人　100109346
弁理士　大貫　敏史
(74)代理人　100117189
弁理士　江口　昭彦
(74)代理人　100134120
弁理士　内藤　和彦

最終頁に続く

(54)【発明の名称】試験システム及びバックアノテーション方法

(57)【特許請求の範囲】
【請求項１】
デバッグによる修正内容を元のソースファイルに反映させるためのバックアノテーション処理機能を有する、デバイスの試験を行うための試験システムであって、
試験を行うためのプログラムを記述したテストプランのソースファイルと、
前記ソースファイルを一以上のブロックに分割した単位で形成される一以上のエレメントと、
前記ソースファイルのオブジェクトをデバッグしたときに、当該デバッグによる修正内容を、当該デバッグの箇所に対応するエレメントに関連付けて管理するアノテータブル・オブジェクトと、
デバッグ後に、前記エレメント及び前記アノテータブル・オブジェクトに基づいて、前記エレメント単位でソースファイルを前記デバッグ後の内容に書き換えるコントローラと、
を備える試験システム。
【請求項２】
前記テストプランは、一以上のソースファイルを備え、
前記一以上のソースファイルのそれぞれについて、前記一以上のエレメントが形成される、
ことを特徴とする請求項１記載の試験システム。
【請求項３】

前記一以上のソースファイルの各ファイル名と対応付けられ、かつ、当該ソースファイルに対応する一連のエレメントを管理するソースファイル情報と、
を備えることを特徴とする請求項２記載の試験システム。
【請求項４】
前記エレメントが形成される単位となるブロックは、前記ソースファイルに記述される分割子に基づいてブロックの区切りが決定される、
ことを特徴とする請求項１乃至３のいずれか１項に記載の試験システム。
【請求項５】
前記デバッグは、ＧＵＩツールを用いて行われる、
ことを特徴とする請求項１乃至４のいずれか１項に記載の試験システム。
【請求項６】
デバイスの試験を行うための試験システムにおいて、デバッグによる修正内容を元のソースファイルに反映させるためのバックアノテーション処理を行う方法であって、
前記試験システムのコントローラが、
試験を行うためのプログラムを記述したテストプランのソースファイルをロードするステップと、
前記ロード後に、前記ソースファイルを一以上のブロックに分割した単位で、一以上のエレメントと形成するステップと、
前記ソースファイルのオブジェクトをデバッグしたときに、当該デバッグによる修正内容を、当該デバッグの箇所に対応するエレメントに関連付けられたアノテータブル・オブジェクトで管理するステップと、
デバッグ後に、前記エレメント及び前記アノテータブル・オブジェクトに基づいて、前記エレメント単位でソースファイルを前記デバッグ後の内容に書き換えるステップと、
を備えるバックアノテーション方法。
【請求項７】
請求項６に記載のバックアノテーション方法をコンピュータに実行させるためのプログラム。
【請求項８】
請求項７に記載のプログラムを記録したコンピュータ読み取り可能な記録媒体。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は自動試験装置の技術分野に関する。特に、本発明は、自動試験装置における試験プログラムのデバッグの際のバックアノテーション処理技術に関する。
【背景技術】
【０００２】
従来、自動試験装置（以下「ＡＴＥ」という。）は、ＡＴＥメーカー各社各様の仕様で提供されていたため、ピン構成や測定ユニット等の構成の自由度が低く、また、試験プログラム資産の再利用が困難であった。このような背景から、デバイスの機能に応じて最適な構成にシステムを変更できるようなスケーラブルで柔軟なＡＴＥを実現すべく、標準化されたインタフェースを持つオープン・アーキテクチャＡＴＥが提案されている。
【０００３】
例えば、ＯＰＥＮＳＴＡＲ（登録商標）は、このようなオープン・アーキテクチャＡＴＥの規格の一つである（非特許文献１参照）。そして、Ｔ２０００テスト・システムは、ＯＰＥＮＳＴＡＲ規格のオープン・アーキテクチャを採用した試験システムである。このＴ２０００テスト・システムでは、デバイス試験のプログラムを、Ｃ＋＋言語のクラスとして記述している（以下、この試験プログラムを「テストプラン」という。）。
【非特許文献１】「Ｓｅｍｉｃｏｎｄｕｃｔｏｒ　Ｔｅｓｔ　Ｃｏｎｓｏｒｔｉｕｍ」、［online］、［平成２０年３月１９日検索］、インターネット＜ＵＲＬ：http://www.semitest.org/jp/home＞

【発明の開示】
【発明が解決しようとする課題】
【０００４】
ところで、ユーザが作成したテストプランをデバッグする際、自動試験装置にテストプランをロードして、ＧＵＩツール等を使用しながらテストプランをデバッグするのが一般的である。ところが、ツールを使用してデバッグをする場合には、テストプラン・オブジェクトに対してデバッグするため、デバッグした内容はそのままでは元のソースファイルに反映されない。そのため、修正内容をソースファイルに反映させるためには、ユーザが自らソースを修正する必要があり、煩雑である。
【０００５】
また、デバッグ後のオブジェクトをリコンパイルして修正済みのソースファイルを得ることも可能ではあるが、この場合には、元のソースファイルで記述した内容が保持されないため、ユーザが記述していたコメント等が破棄されてしまうという問題がある。
【０００６】
本発明は、かかる事情に鑑みてなされたものであり、ＧＵＩツール等を用いてテストプランをデバッグしたとき等に、修正内容を元のソースファイルに自動的に反映させることのできるバックアノテーション処理機能を備えた試験システムを提供することを目的とする。
【課題を解決するための手段】
【０００７】
上記課題を解決するため、本発明に係る試験システムは、デバイスの試験を行うための試験システムであって、試験を行うためのプログラムを記述したテストプランのソースファイルと、ソースファイルを一以上のブロックに分割した単位で形成される一以上のエレメントと、ソースファイルのオブジェクトをデバッグしたときに、当該デバッグによる修正内容を、当該デバッグの箇所に対応するエレメントに関連付けて管理するアノテータブル・オブジェクトと、デバッグ後に、エレメント及びアノテータブル・オブジェクトに基づいて、エレメント単位でソースファイルをデバッグ後の内容に書き換えるコントローラと、を備える。
【０００８】
好適には、テストプランは、一以上のソースファイルを備え、一以上のソースファイルのそれぞれについて、一以上のエレメントが形成される。また、一以上のソースファイルの各ファイル名と対応付けられ、かつ、当該ソースファイルに対応する一連のエレメントを管理するソースファイル情報と、を備えることが好ましい。
【０００９】
また、好適には、エレメントが形成される単位となるブロックは、ソースファイルに記述される分割子に基づいてブロックの区切りが決定される。さらに、デバッグは、ＧＵＩツールを用いて行われることが好ましい。
【００１０】
また、本発明に係るバックアノテーション方法は、デバイスの試験を行うための試験システムにおいて、デバッグによる修正内容を元のソースファイルに反映させるための方法である。ここで、試験システムのコントローラは、試験を行うためのプログラムを記述したテストプランのソースファイルをロードするステップと、ロード後に、ソースファイルを一以上のブロックに分割した単位で、一以上のエレメントと形成するステップと、ソースファイルのオブジェクトをデバッグしたときに、当該デバッグによる修正内容を、当該デバッグの箇所に対応するエレメントに関連付けられたアノテータブル・オブジェクトで管理するステップと、デバッグ後に、エレメント及びアノテータブル・オブジェクトに基づいて、エレメント単位でソースファイルをデバッグ後の内容に書き換えるステップと、を備える。
【００１１】
本発明のプログラムは、本発明に係るバックアノテーション方法の各処理ステップをコ

ンピュータに実行させることを特徴とする。本発明のプログラムは、ＣＤ－ＲＯＭ等の光学ディスク、磁気ディスク、半導体メモリなどの各種の記録媒体を通じて、又は通信ネットワークなどを介してダウンロードすることにより、コンピュータにインストール又はロードすることができる。

【発明の効果】

【００１２】

本発明によると、ツール等を用いてテストプランをデバッグしたとき等に、修正内容を元のソースファイルに自動的に反映させることのできるバックアノテーション処理機能を備えた試験システムを提供することができるようになる。また、本発明によると、デバッグにより修正された箇所のみソースファイルを書き換えることができるため、修正後に得られるソースファイルには、元のソースファイルで記述したコメントが原則保持されるという格別の効果を有する。

【発明を実施するための最良の形態】

【００１３】

以下、本発明の実施の形態について詳細に説明する。なお、同一の要素には同一の符号を付し、重複する説明を省略する。また、以下の実施の形態は、本発明を説明するための例示であり、本発明をその実施の形態のみに限定する趣旨ではない。さらに、本発明は、その要旨を逸脱しない限り、さまざまな変形及び応用が可能である。

【００１４】

図１は、本発明の一実施形態による試験システム１００のシステムアーキテクチャを示す。試験システム１００は、試験信号を生成して被試験デバイス（Ｄｅｖｉｃｅ　ｕｎｄｅｒ　Ｔｅｓｔ。以下「ＤＵＴ」という。）１１２に供給し、ＤＵＴ１１２が試験信号に基づいて動作した結果出力する結果信号が期待値と一致するか否かに基づいてＤＵＴ１１２の良否を判断する。なお、本実施形態に係る試験システム１００は、オープン・アーキテクチャにより実現されるものとして説明するが、本発明は、オープン・アーキテクチャの試験システムに限定されるものではない。

【００１５】

本実施形態では、システムコントローラ（ＳｙｓＣ）１０２がネットワークを介して複数のサイトコントローラ（ＳｉｔｅＣ）１０４に接続される。システムコントローラ１０２は、エンド・ユーザが通常作業を行うホストコンピュータであり、試験システム１００全体を管理する役割を果たす。また、システムコントローラ１０２は、サイトコントローラ１０４に対する処理の要求の発行や、サイトコントローラ１０４間の処理の調停を行う。ユーザのアプリケーションや標準のＧＵＩツールは、システムコントローラ１０２上で動作し、サイトコントローラ１０４と通信を行って機能を実現する。

【００１６】

また、システムコントローラ１０２は、サイトコントローラ１０４上でモジュール１０８の制御を行うモジュール・ソフトウェアや、ユーザの試験プログラムやパターン・プログラムなどを保管するストレージを備える。これらは、必要に応じてサイトコントローラ１０４に送られ、実行される。

【００１７】

各サイトコントローラ１０４は、試験サイト１１０に配置される１つ又は複数のモジュール１０８を制御して試験を実行するために、モジュール接続イネーブラ１０６を通して、モジュール１０８に接続される。この試験は、ユーザの作成する試験プログラムに基づいて実行される。なお、サイトコントローラ１０４は、一つのＤＵＴ１１２を試験する試験サイト１１０を複数個、同時に制御するようにしてもよい。

【００１８】

サイトコントローラ１０４の主な役割として、次の３つが挙げられる。まず、試験プログラムが指定する試験サイト１１０の構成に従い、モジュール接続イネーブラ１０６を構成し、サイトコントローラ１０４とモジュール１０８間のバス１０７の接続を確立する。また、試験サイト１１０内のモジュール１０８の制御を行うモジュール・ソフトウェアを

実行する。さらに、試験プログラムを実行し、各試験サイト１１０のＤＵＴ１１２の試験を実行する。

【００１９】

なお、動作環境によっては、システムコントローラ１０２は、サイトコントローラ１０４の動作とは別のＣＰＵ（中央演算装置）上に配置することができる。別法では、システムコントローラ１０２及びサイトコントローラ１０４は、共通のＣＰＵを共有することができる。同様に、各サイトコントローラ１０４は、その専用のＣＰＵ上に配置することができるし、また、同じＣＰＵ内の別個のプロセス又はスレッドとして配置することもできる。

【００２０】

モジュール接続イネーブラ１０６は、サイトコントローラ１０４とモジュール１０８とのバス１０７の接続を任意に構成することのできるスイッチである。モジュール接続イネーブラ１０６は、接続されるハードウェアモジュール１０８の構成を変更できるようにするとともに、データ転送用（パターンデータのロード用、応答データの収集用、制御用等）のバスとしての役割も果たす。実現可能なハードウェアの実装形態は、専用接続、交換接続、バス接続、リング接続及びスター接続を含む。モジュール接続イネーブラ１０６は、たとえばスイッチマトリクスとして実装することができる。

【００２１】

モジュール１０８は、ＤＵＴ１１２に試験信号を供給する計測機器のハードウェアである。本実施形態においては、モジュール１０８として、オープン・アーキテクチャに基づく各種のモジュールを用いることができる。また、モジュール・ハードウェア１０８を動作させるためのソフトウェアをモジュール・ソフトウェアという。モジュール・ソフトウェアは、デバイス測定時にモジュール１０８の制御をつかさどるモジュール・ドライバ、モジュール１０８の校正と診断を行う校正診断ソフトウェア、モジュール１０８の動作をソフトウェアでエミュレートするエミュレーション・ソフトウェア、モジュール１０８固有のパターン・コンパイラ、およびＧＵＩツールなどを含む。

【００２２】

各試験サイト１１０は、それぞれ一つのＤＵＴ１１２に関連付けられる。ＤＵＴ１１２は、ロードボード１１４を通して、対応する試験サイト１１０のモジュール１０８に接続される。

【００２３】

図２は、試験サイト１１０及びロードボード１１４のハードウェアの概略構成と各種設定用ファイルとの関係の一例を示す図である。図２に示すように、モジュール１０８は、試験システム１００のテストヘッド１３２内のモジュール・スロットに差し込まれる。

【００２４】

試験サイト１１０の構成は、テキスト形式で記述されるソケットファイル１１８で指定される。このソケットファイル１１８には、ＤＵＴ１１２ごとに、ＤＵＴソケット１２０のＤＵＴピン１２２とロードボード１１４上のコネクタピン１２４との接続が記述される。コネクタピン１２４は、ブロック番号とコネクタ番号が付けられており、例えば、「１２．３」のように、「（ブロック番号）．（コネクタ番号）」の形式で表記される。ロードボード１１４のコネクタピン１２４は、テスタ・インタフェース・ユニット（Ｔｅｓｔｅｒ　Ｉｎｔｅｒｆａｃｅ　Ｕｎｉｔ。以下「ＴＩＵ」という。）１２６を介してモジュール１０８のピン１２８と接続される。

【００２５】

一方、ロードボード１１４は試験対象となるデバイスによって異なる場合が多い。そのため、モジュール・ソフトウェアが個々のロードボード１１４の実装に依存しないようにするためには、試験システム１００において、モジュール１０８が実装するピン１２８そのものを定義できるようにし、それをもとにモジュール１０８の制御ができるようにしなければならない。本試験システム１００では、これはモジュール設定ファイル（Ｍｏｄｕｌｅ　Ｃｏｎｆｉｇｕｒａｔｉｏｎ　Ｆｉｌｅ。以下「ＭＣＦ」という。）１３０によっ

て達成される。

【００２６】

ＭＣＦ１３０には、モジュール１０８のリソースとロードボード１１４のコネクタピン１２４との接続関係が指定される。これにより、論理的なピンの表現であるリソースと、モジュール１０８の物理ピン１２８、さらにはそれが接続されているロードボード１１４のコネクタピン１２４との関係が定められる。ＭＣＦ１３０は、システムだけが設定可能であることが望ましい。

【００２７】

なお、本実施形態において、モジュール１０８の機能は、主にモジュール１０８の全体的な制御に関わる機能を提供するモジュール・オブジェクトと、モジュールのピン毎の機能を提供するリソース・オブジェクトによって表現される。本実施形態に係る試験システム１００では、これらは全て、ＭＣＦ１３０に記述される内容に従って、実際のハードウェアのエンティティと対称に結び付けられる。なお、これらのソフトウェアのオブジェクトを提供するモジュール・ソフトウェアを、ここではモジュール・ドライバと呼ぶ。

【００２８】

また、本実施形態において、リソースとは、試験システム１００のモジュール１０８が備える物理的なピン１２８とその機能とを、オブジェクトで抽象化して表現したものである。モジュール１０８の一つの物理ピン１２８は、一つのリソースとして表現されてもよいし、機能的に分割された複数のリソースとして表現されてもよい。

【００２９】

リソースの機能的な分類はリソース・タイプと呼ばれ、その機能の定義はモジュール１０８のベンダが提供するリソース定義ファイルによって示される。論理的なピンを指すリソースは、リソース・タイプと、モジュール１０８内でピン１２８を特定する番号（リソースＩＤ）と、試験システム１００内でモジュール１０８を特定するバス１０７のポート番号との３つ組で表現される。リソース・タイプとリソースＩＤの組と、それに対応するモジュール１０８内の物理ピン１２８の関係は固定的であるため、リソースが与えられたときに、それを物理的なピン１２８に対応付けることが可能となる。なお、リソースが物理ピン１２８と対応付けられることはアーキテクチャ上必須ではなく、モジュール１０８の持つ機能を仮想的なピンに見立てて表現し、制御することも可能である。

【００３０】

図３は、試験システム１００のソフトウェア・アーキテクチャ２００の一例を示す。ソフトウェア・アーキテクチャ２００は分散オペレーティングシステムを表しており、関連するハードウェアシステム要素１０２、１０４、１０８に対応する、システムコントローラ・ソフトウェア２２０（以下、単に「システムコントローラ２２０」ともいう。）、少なくとも１つのサイトコントローラ・ソフトウェア２４０（以下、単に「サイトコントローラ２４０」ともいう。）及び少なくとも１つのモジュールを含むテストヘッド・ソフトウェア２６０（以下、単に「テストヘッド２６０」ともいう。）のための構成要素を有する。

【００３１】

一つの例示的な選択として、試験システム１００は、ソフトウェアのプラットフォームとしてＭｉｃｒｏｓｏｆｔ　Ｗｉｎｄｏｗｓ（登録商標）を使用し、実装言語としてＡＮＳＩ／ＩＳＯ標準Ｃ＋＋を使用することができる。プラットフォームの全てのシステム・インタフェースもまた、Ｃ＋＋のクラスまたはインタフェースとして提供され、ユーザの試験プログラムやモジュール・ソフトウェアはＣ＋＋で実装することができる。

【００３２】

システムコントローラ２２０は、ユーザのための一次的なインタラクションポイントであり、サイトコントローラ２４０へのゲートウェイと、マルチサイト／ＤＵＴ環境におけるサイトコントローラ２４０の同期とを提供する。ユーザアプリケーション及びツールは、グラフィカルユーザインターフェース（ＧＵＩ）に基づくものでも、そうでないものでも、システムコントローラ２２０上で実行される。

【００３３】
また、システムコントローラ２２０は、テストプラン、試験パターン及び試験パラメータファイルを含む、全てのテストプラン関連情報のためのリポジトリとしての役割も果たす。これらのファイルを記憶するメモリは、システムコントローラ２２０に対してローカルに存在することができるか、又は、ネットワークを介してシステムコントローラ２２０に接続されることができる。
【００３４】
さらに、システムコントローラ２２０は、フレームワーククラス２２１と、システムツール２２２と、外部ツール２２３と、サイトコントローラへの標準インタフェース２２４と、通信ライブラリ２３０とを含む。
【００３５】
システムコントローラ２２０に関連するフレームワーククラス２２１は、オブジェクトと対話するための仕組みを提供し、標準インタフェース２２４の参照インプリメンテーションを提供する。また、サイトコントローラ２４０へのゲートウェイを提供し、且つマルチサイト／ＤＵＴ環境におけるサイトコントローラ２４０の同期を提供するソフトウェア構成要素を構成する。実効的には、フレームワーククラス２２１は、システムコントローラ２２０をサポートするＯＳとしての役割を果たすことができる。
【００３６】
第三者の開発者は、標準システムツール２２２に加えて、又は、標準システムツール２２２に換えて、一以上のツール２２３を提供することができる。システムコントローラ２２０上の標準インタフェース２２４は、テスタ及び試験オブジェクトにアクセスするためにそれらのツールが用いるインタフェースを含む。ツール（アプリケーション）２２２、２２３によって、テスタ及び試験オブジェクトをインタラクティブに、且つバッチで制御できる。また、標準インタフェース２２４は、システムコントローラ２２０上で実行されるフレームワークオブジェクトへのオープンインタフェースや、サイトコントローラ２４０に基づくモジュール・ソフトウェアがパターンデータにアクセスし、それらを検索できるようにするインタフェース等を含む。
【００３７】
システムコントローラ２２０上に存在する通信ライブラリ２３０は、ユーザアプリケーション及び試験プログラムに対してトランスペアレントであるように、サイトコントローラ２４０と通信するための仕組みを提供する。
【００３８】
サイトコントローラ２４０は、試験機能の大部分が提供される。サイトコントローラ２４０は、テストプラン（試験プログラム）２４１と、試験クラス２４２と、サイトコントローラフレームワーククラス２４３と、特定モジュール拡張インタフェース２４４と、標準インタフェース２４５と、モジュール・ソフトウェア・インプリメンテーション２４６と、バックプレーン通信ライブラリ２４７と、バックプレーンドライバ２４８とを含む。
【００３９】
テストプラン２４１はユーザによって記述される。テストプランプログラムは、Ｃ＋＋のようなオブジェクト指向コンポーネントを用いて、標準的なコンピュータ言語において直に記述するか、又は、Ｃ＋＋コードを生成するための、さらに高いレベルの試験プログラミング言語において記述することができ、後に実行可能な試験プログラムにコンパイルすることができる。
【００４０】
テストプラン２４１は、標準的な又はユーザによって供給される試験クラス２４２及び／又はそのサイトコントローラ２４０に関連するフレームワーククラス２４３を用いることにより試験オブジェクトを作成し、標準インタフェース２４５を用いてハードウェアを構成し、テストプランフローを規定する。テストプランはいくつかの基本的なサービスをサポートし、デバッグサービス（たとえば、ブレークポイント生成）のような、下層のオブジェクトのサービスへのインタフェースを提供し、下層のフレームワーク及び標準クラ

スにアクセスできるようにする。
【００４１】
試験クラス２４２は、標準試験インタフェースの１つのインプリメンテーションであり、テストプランプログラムにおいて指定される。各試験クラスは通常、特定のタイプのデバイス試験、又はデバイス試験のための設定を実装する。
【００４２】
フレームワーククラス２４３は、共通の試験関連動作を実装する１組のクラス及びメソッドである。フレームワーククラス２４３は、実効的には、各サイトコントローラ２４０をサポートするローカルＯＳとしての役割を果たすことができる。
【００４３】
モジュール・ソフトウェアは、テストの実行時に必要に応じて動的に試験システム１００のプロセスにロードできるように、ＤＬＬ（Ｄｙｎａｍｉｃ　Ｌｉｎｋ　Ｌｉｂｒａｒｙ）形式であることが望ましい。これは、テストプラン２４１が指定する試験サイト１１０の構成に応じて、試験システム１００が実行時に動的に制御対象のモジュール２６１を決定するからである。また、全てのモジュール・ソフトウェアは、モジュール２６１の機能に応じて、システムＯＳが規定するモジュール・ソフトウェアの標準インタフェース２４５を実装することが求められる。
【００４４】
特定モジュール拡張インタフェース２４４は、必要に応じてモジュール固有のより複雑で専門的な機能を持つインタフェース階層をモジュール・ソフトウェアに付加して実装する。例えば、Ｃ＋＋では、標準インタフェース２４５を継承するインタフェースクラスをモジュール・ソフトウェアで定義することによって、インタフェースを拡張することができる。
【００４５】
標準インタフェース２４５は、システムフレームワークが規定する必要最小限の、一般的で普遍的に適用可能なインタフェースである。標準インタフェース２４５を用いて、全てのオブジェクトを画一的に扱うことができる。これにより、システムＯＳを変更することなく、新しいモジュールのソフトウェアをシームレスに、プラグ・アンド・プレイ方式でシステムに導入することが可能となる。
【００４６】
一例として、標準インタフェース２４５は、Ｃ＋＋の純粋仮想インタフェースクラスとして定義される。このとき、ユーザ向けの機能を定義するサブクラスと、システムだけが使用する機能を定義するサブクラスと、リソースの機能を定義するサブクラスとによって構成されることが好ましい。また、標準インタフェース２４５の階層は、モジュールの最も基本的な機能を定義する階層、パターン・プログラムを実行する機能を持つモジュールを表現する階層、複数のピン間で共有されるテスト周期という概念を導入する階層、デジタル・モジュール特有の機能を加える階層、の４階層により構成されることが好ましい。このとき、各モジュール・ドライバは、モジュールの機能に応じて、４階層のインタフェースのいずれかを実装する。しかし、標準インタフェース２４５の構成は、これらの構成に限定されるものではない。
【００４７】
バックプレーン通信ライブラリ２４７は、バックプレーンにわたる標準的な通信のためのインタフェースを提供し、それによりテストヘッド２６０内のモジュール１０８と通信するために必要な機能を提供する。これにより、ベンダ固有のモジュール・ソフトウェアが、バックプレーンドライバ２４８を用いて、対応するモジュール１０８と通信できる。バックプレーン通信プロトコルはパケットに基づくフォーマットを用いることができる。
【００４８】
テストヘッド２６０は、デバイスに対する測定機能が提供される。テストヘッド２６０は、モジュール２６１と、ＴＩＵ２６２と、ロードボード２６３と、ＤＵＴ２６４とを含む。

【００４９】
また、ソフトウェア・アーキテクチャ２００は、ソフトウェアの名目的な供給元に基づいて、システムフレームワーク２９０、ユーザコンポーネント２９２、テスタオペレーティングシステム２９４、モジュール・ハードウェア・ベンダコンポーネント２９６、及び、ツール・ソフトウェア・ベンダコンポーネント２９８に分類することができる。
【００５０】
システムフレームワーク２９０は、試験システム１００の開発ベンダにより供給され、フレームワーククラス２２１と、標準インタフェース２２４と、フレームワーククラス２４３と、標準インタフェース２４５と、バックプレーン通信ライブラリ２４７とを含む。
【００５１】
ユーザコンポーネント２９２は、試験を実施するユーザによって供給され、テストプラン２４１と、テストクラス２４２と、ロードボード２６３と、ＤＵＴ２６４とを含む。
【００５２】
テスタオペレーティングシステム２９４は、基本的な接続性及び通信のためのソフトウェアインフラストラクチャとして供給され、システムツール２２２と、通信ライブラリ２３０と、バックプレーンドライバ２４８とを含む。
【００５３】
モジュール・ハードウェア・ベンダコンポーネント２９６は、モジュール１０８の開発元によって供給され、特定モジュール拡張インタフェース２４４と、モジュール・ソフトウェア・インプリメンテーション２４６と、モジュール２６１と、ＴＩＵ２６２を含む。
【００５４】
ツール・ソフトウェア・ベンダコンポーネント２９８は、外部ツールの開発元によって供給され、外部ツール２２３を含む。
【００５５】
次に、以上のように構成される試験システム１００における、バックアノテーション処理について説明する。ここで、バックアノテーションとは、ＧＵＩツール（ツール２２２，２２３）等を使ってデバッグしたテストプランやパターンデータ等の修正内容を、テストプランやパターンデータ等のソースファイルに反映させる機能をいう。なお、バックアノテーションは、汎用デバッガによりソースコードを直接開いて行うデバッグとは異なる。汎用デバッガの場合、ソースを直接開いてプログラムをデバッグするため、そのソースを保存するだけでデバッグの結果を反映させることができる。一方、バックアノテーションの場合、デバッグの対象はデータであり、データ毎にデバッグ用のツールが用意されているのであって、元のソースを直接修正するものではない。
【００５６】
例えば、テストプランを開発するとき、ユーザはＧＵＩツール等を使ってテストプランのフロー、テスト条件、パターンデータなどを修正しながらデバッグする。バックアノテーション機能を使用しない場合、種々のパラメータの修正を、手作業でテストプランやパターンデータのソースファイルに反映させなければならない。一方、バックアノテーション機能を使用すると、パラメータの修正内容をテストプランやパターンデータのソースファイルに自動的に反映させて保存できる。そして、保存されたソースファイルをコンパイル、ロードすることにより、デバッグ結果が反映されたテストプランが実行できる。本実施形態に係るバックアノテーション機能を使用したとき、ソースファイルに記述されたコメントや、ＧＵＩツールで修正できないパラメータの値は、元の修正前の状態のまま保持される。修正内容としては、ユーザ変数の値、スペシフィケーション・セット、テスト条件、リソース・パラメータの設定、タイミングの設定、フロー・アイテム、テスト内容等の追加、削除、及び実行順序の変更などがある。
【００５７】
また、パターンデータのバックアノテーションの場合、パターン・エディタを使ってデバッグしたパターン・プログラムの修正内容を、パターン・オブジェクトのメタファイルに反映させる。例えば、パターン・プログラムを開発するとき、ユーザはパターン・エデ

ィタを使用して、パターン・ジェネレータにロードされたパターン・プログラムを修正しながらデバッグする。バックアノテーションを使用しない場合、パターン・プログラムの修正を、手作業でパターン・プログラムのソースファイルに反映させなければならず、デバッグ済みのパターン・オブジェクトのメタファイルを得るには、さらに、修正したパターン・ソースファイルをコンパイルする必要がある。一方、バックアノテーションを使用すると、デバッグしたパターン・プログラムをパターン・オブジェクトのメタファイルに保存できる。保存されたパターン・オブジェクトのメタファイルを再ロードすることにより、デバッグ結果が反映されたパターン・プログラムを実行できる。

【００５８】

バックアノテーションを使用することにより、テストプランやパターン・プログラムを開発するときの、ターン・アラウンド・タイムを短縮できるが、本発明は特に、テストプランのバックアノテーション処理に特徴を有するものである。すなわち、ユーザがロードされたテストプランのランタイム・オブジェクトを修正すると、修正された情報は、バックアノテーション機能によって、テストプラン言語で記述されたソースに反映された後、保存される。

【００５９】

なお、バックアノテーションを実現するためには、新規ソース保存方式と、差分保存方式と、オリジナルソース保存方式に分けられる。新規ソース保存方式は、新たなテストプランを記述したファイルとして、テストプランのオブジェクトを新たに生成して保存するが、元のソースファイル内のコメント、余白、順序等は失われる。差分保存方式は、実行しているランタイム・オブジェクトに加えた変更の差分を保存する。この方式の場合、修正されたテストプランを取得するためには、変更の差分に沿って、元のテストプランをロードする必要がある。また、オリジナルソース保存方式では、オリジナルのソースファイルに変更を加える。この方式の場合、コメントや余白が維持される。本発明は、オリジナルソース保存方式に基づいて、バックアノテーションを実現するものである。

【００６０】

以下、本発明に係るバックアノテーション処理について説明する。

【００６１】

図４は、本試験システム１００において、テストプランのバックアノテーション処理を行うためのデータ構造を示す図である。図４に示すように、バックアノテーション処理のデータ構造１０は、テストプランを構成する各ソースファイル名１２とそのソースファイル情報１４とを対応付けるマップ１６を含んでいる。各ソースファイル情報１４はそれぞれ、ソースファイルを所定単位で分割したエレメント２０のリストと関連付けられている。そして、テストプランのランタイム・オブジェクトが修正されたとき、修正された箇所に対応するエレメント２０に関連付けられるアノテータブル・オブジェクト１８が形成され、当該修正された内容が管理される。なお、テストプランは一以上のソースファイルにより構成され、各ソースファイルには、試験を行うためのプログラムが記述されている。

【００６２】

データ構造１０は、ユーザがバックアノテーション・タスクを開始するファイル保存オペレーションを呼び出すことによって生成される。

【００６３】

ソースファイル名１２は、システムにロードされたテストプランのソースファイルの名称である。本試験システム１００では、サイトコントローラ１０４がテストプランをロードし、サイトコントローラ１０４上でバックアノテーションを実施するためのデータ構造１０が構築される。

【００６４】

ソースファイル情報１４は、対応するソースファイル名１２に関する情報と、ネットワークのファイルパスと、関連付けられるエレメント２０のリストと、を含んでいる。これにより、テストプランを構成する一以上のソースファイル名１２と対応付けられ、かつ、当該ソースファイルに対応する一連のエレメント２０を管理する。このエレメント２０の

リストは、エレメント２０へのポインタにより構成される。このように、テストプラン内のオブジェクトはオブジェクトが宣言されたエレメントへのポインタを有している。これらのポインタは、エレメントが挿入されたり削除されたりしても、無効とされてはならないので、エレメント２０が存続している間は、ポインタが制御される。

【００６５】

マップ１６は、システムにロードされたテストプランに関する一以上のソースファイル名１２と、当該ソースファイルのソースファイル情報１４とを対応付けを示すものである。

【００６６】

アノテータブル・オブジェクト１８は、個々のエレメント２０と一対一に対応するオブジェクトであり、アノテータブルクラスは、フレームワーククラスと対になって用意され、ブロックごとにオブジェクトが作成される。デバッグによりデータに変更が加えられると、当該変更内容を管理し、バックアノテーションを実施するために必要なオペレーションを特定する。このオペレーションは、オブジェクトが修正されたか否かを判断し、修正された場合には、エレメント２０に修正されたことを示す所定のマークを付けたり、修正内容に基づいてエレメント２０のテキストを表現したりする等の処理を含む。また、アノテータブル・オブジェクト１８は、必要な場合に、エレメント２０のテキストを表現するためのメソッドの実装を提供する。バックアノテーション実行時に、フレームワーククラスから最新のデータを取得して、ソースファイルに書き戻すためのテキストデータを作成する。

【００６７】

エレメント２０は、テストプランのソースファイルを一以上のブロックに分割した単位で形成され、プログラムの記述位置などを管理するソースファイル情報１４と関連付けられる。これにより、ソースファイルは、一続きのエレメント２０のリストとして表される。そして、ソースファイルを構成するブロックごとやブロック内に記述する要素ごとにエレメント２０が作成され、対応するデータに変更が加えられたとき、その変更内容をソースファイルのどの位置に書き戻せばよいかを示す。本実施形態において、エレメントが形成される単位となるブロックは、ソースファイルに記述される“；”、“｝”等の分割子に基づいてブロックの区切りが決定される。ソースファイル内の宣言に関する情報が捕捉され、ネストの無い単純なオブジェクトの宣言が、一つのエレメント２０として表される。また、ネストされた宣言を有するオブジェクトの宣言は、一つのエレメント内にヘッダ又はフッタを有するエレメント２０として表される。エレメント２０は、例えば、宣言されたオブジェクトの名称と、アイテムテキスト長と、アノテータブル・オブジェクト１８へのポインタとを備える。そして、バックアノテーション処理においては、デバッグによりテストプランが修正されたとき、エレメント単位でソースファイルが修正される。

【００６８】

図５は、エレメント２０の一例を示す図である。図５（Ａ）はテストプランのソースファイルの一例である。そして、図５（Ｂ）は、図５（Ａ）に示すソースファイルに基づいて形成されるエレメント２０を示す図である。

【００６９】

図５（Ｂ）に示すように、この例におけるソースファイルは、５つのエレメント２１，２２，２３，２４，２５に分割される。エレメント２１は、宣言ＤＵＴＴｙｐｅに関するエレメントである。エレメント２２は、宣言ＳｐｅｃｉｆｉｃａｔｉｏｎＳｅｔに関するエレメントである。エレメント２３は、宣言Ｖｏｌｔａｇｅに関するエレメントである。エレメント２４は、宣言Ｃｕｒｒｅｎｔに関するエレメントである。エレメント２５は、宣言ＳｐｅｃｉｆｉｃａｔｉｏｎＳｅｔの後書きエレメントである。

【００７０】

ソースファイルがコメントや改行を含むとき、通常これらのコメントや改行のみでは、エレメントを形成しない。コメントや改行等は、これらに続いて現れる宣言に関するエレメントに関連付けられる。例えば、エレメント２２に示すように、エレメントのヘッダと

して構成される。また、エレメント２０が入れ子状に構成されることはない。ネストされた宣言を有するオブジェクトは、エレメントのヘッダ又はフッタとして表現される。

【００７１】

あるエレメント２０が修正されても、そのコメントや改行は保持される。例えば、エレメント２２のＳｐｅｃｉｆｉｃａｔｉｏｎＳｅｔが修正されても、エレメント２２内のコメント１～３は変化しない。同様に、エレメント２３のＶｏｌｔａｇｅ　Ｖの値が１．０から１．１に修正されても、エレメント２３内のコメント４は変化しない。

【００７２】

しかしながら、エレメント２３のＶｏｌｔａｇｅ　Ｖの値が１．０から１．１に修正された場合、内部コメント１及び２は保持されずに破壊されてしまう。そして、エレメント２３のＶｏｌｔａｇｅ　Ｖの宣言は、次のようにソースファイルに書き戻されて修正される。

Ｖｏｌｔａｇｅ　Ｖ　＝　１．１，２．０，１．５；

【００７３】

ソースファイル内のコメントは、当該コメントに続くエレメントに関連付けられる。そして、エレメントと同じ文法レベルのコメントは保持される。しかし、宣言に対してネストされたレベルのコメント、すなわち、例えばエレメント２３内の内部コメント１及び内部コメント２は、表現が修正されたとき保持されない。また、各エレメントの最後のセミコロンの後にコメントがある場合、当該コメントは、次のエレメントに含まれることになる。また、エレメント２０は、ファイルの名称を備えることが好ましい。

【００７４】

図６は、試験システム１００においてテストプランのデバッグをする際の、バックアノテーション処理の流れを示すフローチャートである。

【００７５】

まず、テストプランがサイトコントローラ１０４上でロードされる（Ｓ３１）。このとき、ソースファイルのブロックごとに、そのブロック内に記述したデータを管理するためのフレームワーククラスが用意されている。例えば、ＵｓｅｒＶａｒｓブロックの記述内容を管理するため、クレームワーククラスの一つであるＵｓｅｒＶａｒｓクラスのオブジェクトが作成される。ユーザが、バックアノテーション・タスクを開始するファイル保存オペレーションを呼び出している場合には、テストプランのロード時にコードが生成され、ソースファイル名１２からソースファイル情報１４へのマップ１６が形成される。また、ソースファイル情報１４に関連付けられる一連のエレメント２０が生成される（Ｓ３２）。なお、ロードされたテストプランのオブジェクトは、マップ１６と、対応するエレメント２０へのポインタを含む。

【００７６】

ここで、ユーザは、ＧＵＩツール等を使用しながら、テストプランやパターンデータのデバッグを行う（Ｓ３３）。このＧＵＩツール等を用いたデバッグにより、データが編集されたり、削除されたり、新たに挿入されたりしてデータに変更が加えられると、変更された箇所に対応するアノテータブル・オブジェクト１８に、その変更内容が管理される（Ｓ３４）。

【００７７】

例えば、ユーザ変数の初期値を変更するなどして、オブジェクトが編集されたとき、編集された箇所を示すエレメントに、編集されたことを示すマークが付される。また、データが削除されたとき、削除された箇所に対応するエレメントに削除されたことを示すマークが付される。このとき、当該エレメントに結び付けられたコメントも削除される。データが新たに挿入されたとき、新たに挿入されたデータに対応するエレメント２０を新たに生成し、当該エレメント２０にて、新たに挿入されたオブジェクトに関するデータが管理される。

【００７８】

そして、デバッグ後にバックアノテーションが要求されたとき、エレメント２０により

変更された箇所を特定し、エレメント２０が示すソースファイルの位置のテキストを、アノテータブル・オブジェクト１８から取得した変更後のテキストデータに書き換える（Ｓ３５）。例えば、編集されたことを示すマークが付されたエレメント２０については、アノテータブル・オブジェクト１８の内容に基づいて、ソースファイルが書き換えられる。このとき、ソースファイル中のコメント等は書き換えられずに保持される。なお、生成されるソースは、言語モジュールを使って、テストプラン・オブジェクトから生成される。

【００７９】

なお、テストプランのデバッグが済んで実際にＤＵＴ１１２の試験を行う場合など、テストプランを修正しない場合には、テストプランをロードしたときに、ステップＳ３２に示すようなエレメント２０等の生成処理は不要である。

【００８０】

また、バックアノテーション機能は、テストプランＤＬＬからテストプログラムをロードした場合、またはテストプラン・ローダを使用してソースファイルから直接テストプログラムをロードした場合のいずれでも使用可能である。

【００８１】

なお、本発明は、上記した実施の形態に限定されるものではなく、本発明の要旨を逸脱しない範囲内において、他の様々な形で実施することができる。このため、上記実施形態はあらゆる点で単なる例示にすぎず、限定的に解釈されるものではない。例えば、上述の各処理ステップは処理内容に矛盾を生じない範囲で任意に順番を変更して又は並列に実行することができる。

【図面の簡単な説明】

【００８２】

【図１】本発明の一実施形態による試験システム１００のシステムアーキテクチャを示す。

【図２】試験サイト１１０及びロードボード１１４のハードウェアの概略構成と各種設定用ファイルとの関係の一例を示す図である。

【図３】試験システム１００のソフトウェア・アーキテクチャ２００の一例を示す。

【図４】テストプランのバックアノテーション処理を行うためのデータ構造を示す図である。

【図５】エレメント２０の一例を示す図である。

【図６】バックアノテーション処理の流れを示すフローチャートである。

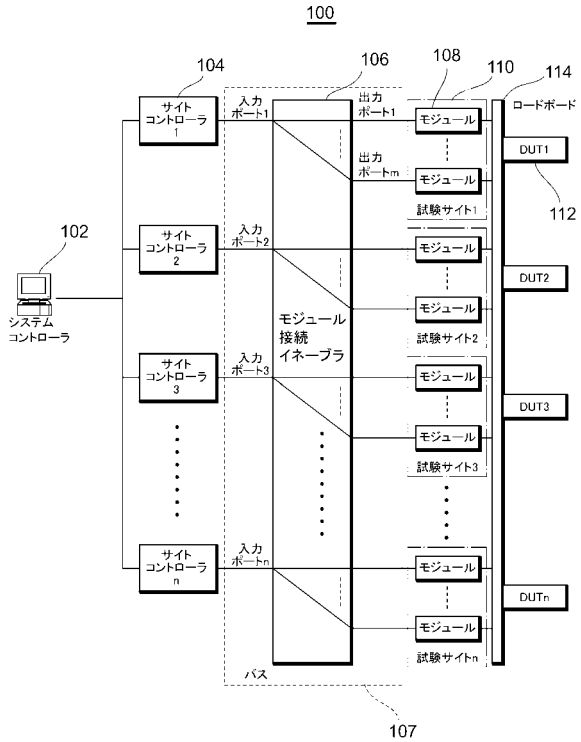
【符号の説明】

【００８３】

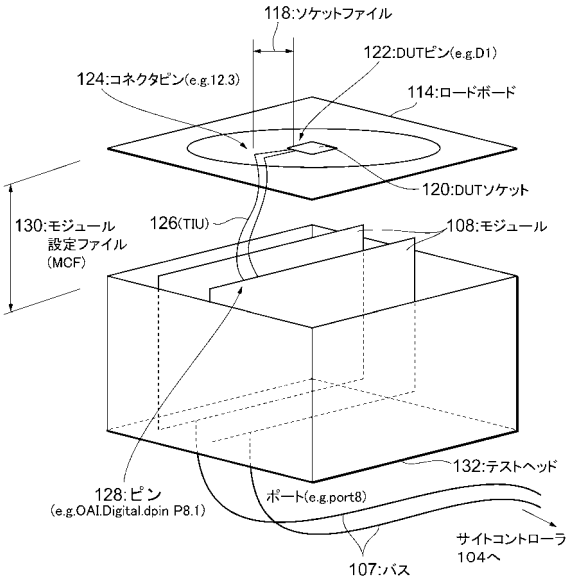
１０　データ構造
１２　ソースファイル名
１４　ソースファイル情報
１６　マップ
１８　アノテータブル・オブジェクト
２０　エレメント
１００ 試験システム
１０２ システムコントローラ
１０４ サイトコントローラ
１０６ モジュール接続イネーブラ
１０７ バス
１０８ モジュール
１１０ 試験サイト
１１２ 被試験デバイス（ＤＵＴ）
１１４ ロードボード
１１８ ソケットファイル
１２０ ソケット

１２２ ピン
１２４ コネクタピン
１２６ テスタ・インタフェース・ユニット（ＴＩＵ）
１２８ ピン
１３０ モジュール設定ファイル（ＭＣＦ）
１３２ テストヘッド

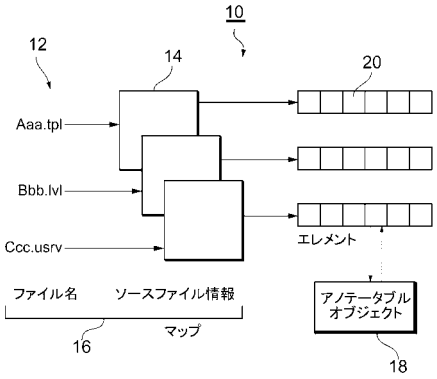
【図１】



【図２】



【図４】

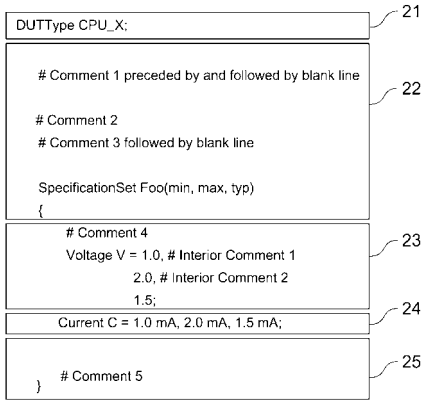


【図５】

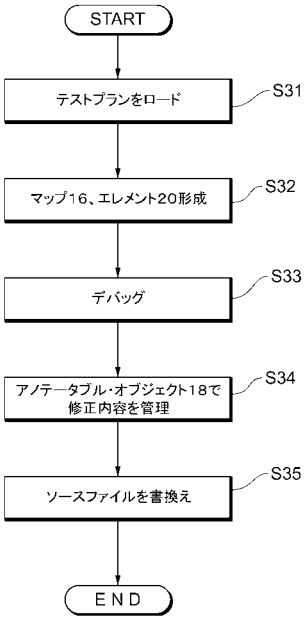
(A)

```
DUTType CPU_X;

    # Comment 1 preceded by and followed by blank line

    # Comment 2
    # Comment 3 followed by blank line

    SpecificationSet Foo(min, max, typ)
    {
        # Comment 4
        Voltage V = 1.0, # Interior Comment 1
                    2.0, # Interior Comment 2
                    1.5;
        Current C = 1.0 mA, 2.0 mA, 1.5 mA;

        # Comment 5
    }
```

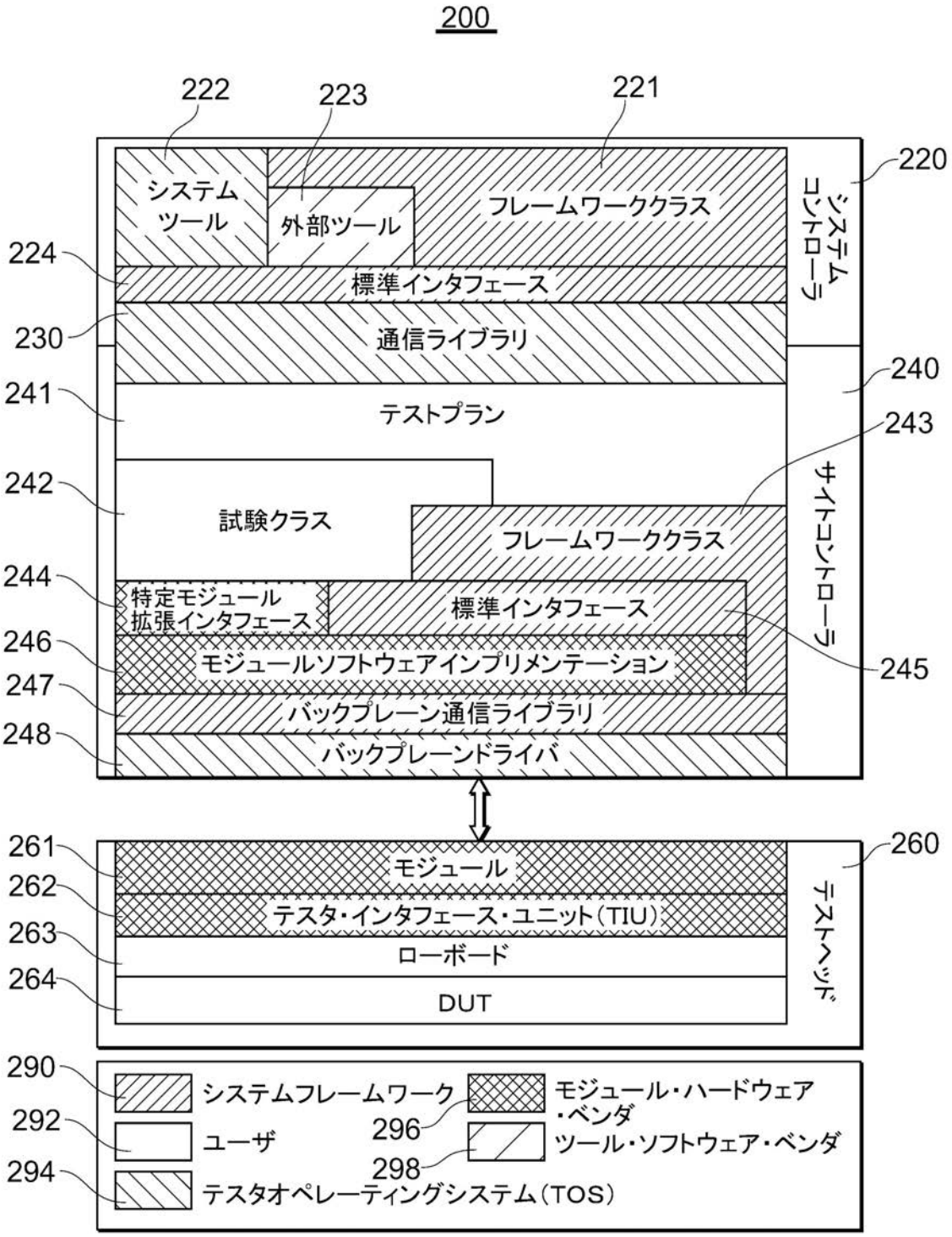
(B)



【図６】



【図３】


200
222
223
221
220
システム
ツール
外部ツール
フレームワーククラス
システム
コントローラ
224
標準インタフェース
230
通信ライブラリ
240
241
テストプラン
243
242
試験クラス
フレームワーククラス
サイトコントローラ
244
特定モジュール
拡張インタフェース
標準インタフェース
246
モジュールソフトウェアインプリメンテーション
245
247
バックプレーン通信ライブラリ
248
バックプレーンドライバ
261
260
モジュール
262
テスタ・インタフェース・ユニット（TIU）
263
ローボード
264
DUT
テストヘッド
290
システムフレームワーク
モジュール・ハードウェア
・ベンダ
292
296
ユーザ
ツール・ソフトウェア・ベンダ
298
294
テスタオペレーティングシステム（TOS）

フロントページの続き

(72)発明者　横山　優
東京都練馬区旭町１－３２－１　株式会社アドバンテスト内

審査官　▲高▼橋　正▲徳▼

(56)参考文献　特開平０５－１３４０１２（ＪＰ，Ａ）
特開平０５－３２４３０２（ＪＰ，Ａ）
特開２００７－２６５０８９（ＪＰ，Ａ）
特表２００７－５２８９９４（ＪＰ，Ａ）
特開平０８－００６８２４（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　１１／２２－１１／２６，
Ｇ０１Ｒ　３１／２８－３１／３０，
Ｇ０６Ｆ　９／０６