(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)特許出願公開番号
特開2006-244451
(P2006-244451A)
(43)公開日　平成18年9月14日(2006.9.14)

(51)Int.Cl.		F I		テーマコード（参考）
G06F 9/50	(2006.01)	G06F 9/46	462Z	5B042
G06F 11/34	(2006.01)	G06F 11/34	S	

審査請求 有　請求項の数 18　ＯＬ　（全 49 頁）

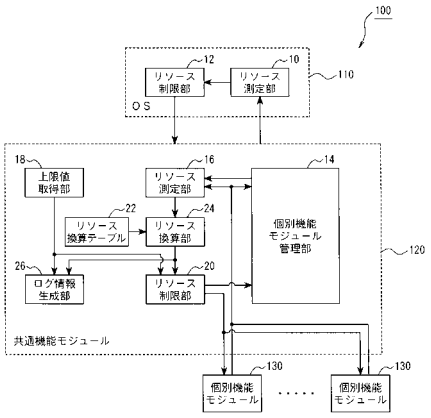
(21)出願番号　特願2005-244162 (P2005-244162)
(22)出願日　平成17年8月25日 (2005.8.25)
(31)優先権主張番号　特願2005-24983 (P2005-24983)
(32)優先日　平成17年2月1日 (2005.2.1)
(33)優先権主張国　日本国 (JP)

(71)出願人　000002369
セイコーエプソン株式会社
東京都新宿区西新宿２丁目４番１号
(74)代理人　100066980
弁理士　森　哲也
(74)代理人　100075579
弁理士　内藤　嘉昭
(74)代理人　100103850
弁理士　崔　秀▲てつ▼
(72)発明者　谷口　真也
長野県諏訪市大和３丁目３番５号　セイコーエプソン株式会社内
(72)発明者　深尾　明人
長野県諏訪市大和３丁目３番５号　セイコーエプソン株式会社内

最終頁に続く

(54)【発明の名称】リソース管理システムおよびリソース管理プログラム、並びにリソース管理方法

(57)【要約】
【課題】　ソフトウェアの開発を容易にするとともに安定性の高いソフトウェアを開発するのに好適なリソース管理システムを提供する。
【解決手段】　ホスト端末１００は、個別機能モジュール１３０がホスト端末１００で使用するリソースの量を測定し、測定したリソースの量を、ネットワークプリンタで使用するリソースの量に換算し、個別機能モジュール１３０から上限値を取得し、換算したリソースの量および取得した上限値に基づいて、個別機能モジュール１３０が使用するリソースの量が上限に達したことを示すログ情報を生成する。また、各個別機能モジュール１３０ごとに、換算したリソースの量および取得した上限値に基づいて、その個別機能モジュール１３０が使用するメモリ量、並びに共通機能モジュール１２０がその個別機能モジュール１３０の実行に使用するメモリ量および起動するクラス数を制限する。
【選択図】　図２



【特許請求の範囲】
【請求項１】
機能モジュールが使用するリソースを管理するリソース管理システムであって、
前記機能モジュールが第１実行環境で使用するリソースの量を測定するリソース測定手段と、第２実行環境でのリソースの上限値を取得するリソース上限値取得手段と、前記リソース上限値取得手段で取得したリソースの上限値を、前記第１実行環境でのリソースの上限値に換算するリソース換算手段と、前記リソース測定手段で測定したリソースの量および前記リソース換算手段で換算したリソースの上限値に基づいてリソースの制限に関する通知を行うリソース制限通知手段とを備えることを特徴とするリソース管理システム。
【請求項２】
第１機能モジュールと、その実行に前記第１機能モジュールを必要とする複数の第２機能モジュールとがが使用するリソースを管理するリソース管理システムであって、
前記第２機能モジュールが第１実行環境で使用するリソースの量を測定するリソース測定手段と、第２実行環境でのリソースの上限値を取得するリソース上限値取得手段と、前記リソース上限値取得手段で取得したリソースの上限値を、前記第１実行環境でのリソースの上限値に換算するリソース換算手段と、前記リソース測定手段で測定したリソースの量および前記リソース換算手段で換算したリソースの上限値に基づいてリソースの制限に関する通知を行うリソース制限通知手段とを備えることを特徴とするリソース管理システム。
【請求項３】
請求項２において、
前記リソース測定手段は、前記第２機能モジュールが前記第１実行環境で使用するリソースの量、および前記第１機能モジュールが前記第１実行環境で当該第２機能モジュールの実行に使用するリソースの量を測定するようになっていることを特徴とするリソース管理システム。
【請求項４】
第１機能モジュールと、その実行に前記第１機能モジュールを必要とする複数の第２機能モジュールとが使用するリソースを管理するリソース管理システムであって、
前記第１機能モジュールが前記第１実行環境で前記第２機能モジュールの実行に使用するリソースの量を測定するリソース測定手段と、第２実行環境でのリソースの上限値を取得するリソース上限値取得手段と、前記リソース上限値取得手段で取得したリソースの上限値を、前記第１実行環境でのリソースの上限値に換算するリソース換算手段と、前記リソース測定手段で測定したリソースの量および前記リソース換算手段で換算したリソースの上限値に基づいてリソースの制限に関する通知を行うリソース制限通知手段とを備えることを特徴とするリソース管理システム。
【請求項５】
請求項２ないし４のいずれか１項において、
前記リソース測定手段で測定したリソースの量および前記リソース換算手段で換算したリソースの上限値に基づいて前記機能モジュールによるリソースの確保を禁止するリソース制限手段を備えることを特徴とするリソース管理システム。
【請求項６】
請求項１ないし５のいずれか１項において、
前記リソース換算手段は、前記第１実行環境で所定条件で使用されるリソースの量および前記第２実行環境で前記所定条件で使用されるリソースの量に基づき決定される換算率を登録したリソース換算テーブルに基づいて換算を行うようになっていることを特徴とするリソース管理システム。
【請求項７】
請求項６において、
前記リソース換算テーブルは、リソースの種別ごとに前記換算率を登録したものであり、

前記リソース換算手段は、前記機能モジュールが使用するリソースの種別に応じて前記リソース換算テーブルから対応する前記換算率を取得し、取得した換算率に基づいて換算を行うようになっていることを特徴とするリソース管理システム。

【請求項８】

請求項６および７のいずれか１項において、

前記リソース換算テーブルは、リソースの使用形態ごとに前記換算率を登録したものであり、

前記リソース換算手段は、前記機能モジュールが使用するリソースの形態に応じて前記リソース換算テーブルから対応する前記換算率を取得し、取得した換算率に基づいて換算を行うようになっていることを特徴とするリソース管理システム。

【請求項９】

請求項６ないし８のいずれか１項において、

前記リソース換算テーブルは、複数のテストモジュールのそれぞれについて、前記第１実行環境および前記第２実行環境で当該テストモジュールが使用するリソースの量に基づいて前記換算率が決定され、前記各テストモジュールについて決定された前記換算率のうち最大値を登録したものであることを特徴とするリソース管理システム。

【請求項１０】

請求項６ないし８のいずれか１項において、

前記リソース換算テーブルは、複数のテストモジュールのそれぞれについて、前記第１実行環境および前記第２実行環境で当該テストモジュールが使用するリソースの量に基づいて前記換算率が決定され、前記各テストモジュールについて決定された前記換算率の平均値を登録したものであることを特徴とするリソース管理システム。

【請求項１１】

請求項１ないし１０のいずれか１項において、

前記リソース制限通知手段は、前記機能モジュールが使用するリソースの量が上限に達したことを示すメッセージを表示するようになっていることを特徴とするリソース管理システム。

【請求項１２】

請求項１ないし１０のいずれか１項において、

前記リソース制限通知手段は、前記機能モジュールが使用するリソースの量が上限に達したことを示すログ情報を生成するようになっていることを特徴とするリソース管理システム。

【請求項１３】

機能モジュールが使用するリソースを管理するリソース管理プログラムであって、

前記機能モジュールが第１実行環境で使用するリソースの量を測定するリソース測定ステップと、第２実行環境でのリソースの上限値を取得するリソース上限値取得ステップと、前記リソース上限値取得ステップで取得したリソースの上限値を、前記第１実行環境でのリソースの上限値に換算するリソース換算ステップと、前記リソース測定ステップで測定したリソースの量および前記リソース換算ステップで換算したリソースの上限値に基づいてリソースの制限に関する通知を行うリソース制限通知ステップとからなる処理をコンピュータに実行させるためのプログラムを含むことを特徴とするリソース管理プログラム。

【請求項１４】

第１機能モジュールと、その実行に前記第１機能モジュールを必要とする複数の第２機能モジュールとが使用するリソースを管理するリソース管理プログラムであって、

前記第２機能モジュールが第１実行環境で使用するリソースの量を測定するリソース測定ステップと、第２実行環境でのリソースの上限値を取得するリソース上限値取得ステップと、前記リソース上限値取得ステップで取得したリソースの上限値を、前記第１実行環境でのリソースの上限値に換算するリソース換算ステップと、前記リソース測定ステップで測定したリソースの量および前記リソース換算ステップで換算したリソースの上限値に

基づいてリソースの制限に関する通知を行うリソース制限通知ステップとからなる処理をコンピュータに実行させるためのプログラムを含むことを特徴とするリソース管理プログラム。

【請求項１５】

第１機能モジュールと、その実行に前記第１機能モジュールを必要とする複数の第２機能モジュールとが使用するリソースを管理するリソース管理プログラムであって、

前記第１機能モジュールが前記第１実行環境で前記第２機能モジュールの実行に使用するリソースの量を測定するリソース測定ステップと、第２実行環境でのリソースの上限値を取得するリソース上限値取得ステップと、前記リソース上限値取得ステップで取得したリソースの上限値を、前記第１実行環境でのリソースの上限値に換算するリソース換算ステップと、前記リソース測定ステップで測定したリソースの量および前記リソース換算ステップで換算したリソースの上限値に基づいてリソースの制限に関する通知を行うリソース制限通知ステップとからなる処理をコンピュータに実行させるためのプログラムを含むことを特徴とするリソース管理プログラム。

【請求項１６】

機能モジュールが使用するリソースを管理するリソース管理方法であって、

前記機能モジュールが第１実行環境で使用するリソースの量を測定するリソース測定ステップと、第２実行環境でのリソースの上限値を取得するリソース上限値取得ステップと、前記リソース上限値取得ステップで取得したリソースの上限値を、前記第１実行環境でのリソースの上限値に換算するリソース換算ステップと、前記リソース測定ステップで測定したリソースの量および前記リソース換算ステップで換算したリソースの上限値に基づいてリソースの制限に関する通知を行うリソース制限通知ステップとを含むことを特徴とするリソース管理方法。

【請求項１７】

第１機能モジュールと、その実行に前記第１機能モジュールを必要とする複数の第２機能モジュールとが使用するリソースを管理するリソース管理方法であって、

前記第２機能モジュールが第１実行環境で使用するリソースの量を測定するリソース測定ステップと、第２実行環境でのリソースの上限値を取得するリソース上限値取得ステップと、前記リソース上限値取得ステップで取得したリソースの上限値を、前記第１実行環境でのリソースの上限値に換算するリソース換算ステップと、前記リソース測定ステップで測定したリソースの量および前記リソース換算ステップで換算したリソースの上限値に基づいてリソースの制限に関する通知を行うリソース制限通知ステップとを含むことを特徴とするリソース管理方法。

【請求項１８】

第１機能モジュールと、その実行に前記第１機能モジュールを必要とする複数の第２機能モジュールとが使用するリソースを管理するリソース管理方法であって、

前記第１機能モジュールが前記第１実行環境で前記第２機能モジュールの実行に使用するリソースの量を測定するリソース測定ステップと、第２実行環境でのリソースの上限値を取得するリソース上限値取得ステップと、前記リソース上限値取得ステップで取得したリソースの上限値を、前記第１実行環境でのリソースの上限値に換算するリソース換算ステップと、前記リソース測定ステップで測定したリソースの量および前記リソース換算ステップで換算したリソースの上限値に基づいてリソースの制限に関する通知を行うリソース制限通知ステップとを含むことを特徴とするリソース管理方法。

【発明の詳細な説明】

【技術分野】

【０００１】

本発明は、リソースを管理するシステムおよびプログラム、並びに方法に係り、特に、ソフトウェアがその実行環境で使用するリソースの量をその実行環境への導入前に検証することにより、ソフトウェアの開発を容易にするとともに安定性の高いソフトウェアを開発するのに好適なリソース管理システムおよびリソース管理プログラム、並びにリソース

管理方法に関する。
【背景技術】
【０００２】
プリンタ等の組込機器には、組込用アプリケーションと呼ばれるソフトウェアを組み込んでその動作を制御している。しかしながら、組込用アプリケーションを作成するには、一般に専用の開発環境と専用のハードウェアを必要とするため、誰もが簡単に作成できるものではなかった。この問題を解決するために、特許文献１記載の情報処理装置が提案されている。
【０００３】
特許文献１記載の発明は、画像形成装置上で実行されるアプリケーションをＰＣ上で実行できるエミュレータを有して構成されている。これにより、組込機器を用いなくても組込用アプリケーションを開発することができる。
また、プログラミング技術が未熟な人が作成する組込用アプリケーションは、予期せぬ動作を引き起こし、組込機器自体の動作が続行できなくなるという問題がある。この問題を解決するために、特許文献２記載のリソース管理システムが提案されている。
【０００４】
特許文献２記載の発明は、ソフトウェアが情報機器で実行するときに利用するリソースに対して動作可能な範囲を設定する制限設定部と、制限設定部で設定した動作可能な範囲内で動作していることを検証する動作範囲検証部とを有して構成されている。動作範囲検証部は、ソフトウェアからリソース利用要求があったときに、設定してある動作可能な範囲と要求されたリソースの量とを比較し、動作可能な範囲外のときにはそのソフトウェアの実行を中止させる。
【特許文献１】特開２００４－１８５５９５号公報
【特許文献２】特開２００４－９４７８２号公報
【発明の開示】
【発明が解決しようとする課題】
【０００５】
特許文献１記載の発明では、組込用アプリケーションをＰＣ上で仮想的に実行することができるため、組込用アプリケーションの動作をＰＣ上である程度検証することができる。
しかしながら、組込用アプリケーションをＰＣ上で実行した場合と、実際にその組込用アプリエーションを組込機器上で実行した場合とでは、その組込用アプリエーションがそれぞれの実行環境で使用するリソース（例えば、メモリ）の量が完全には一致しない。これは、組込用アプリケーションを実行する環境が異なることに起因する。例えば、組込用アプリケーションを実行するために、ライブラリを使用するが、このライブラリは、ハードウェア構成の違いからＰＣ用と組込機器用とでそれぞれプログラム構造が異なる。そのため、同じ関数を使用する場合でも、ＰＣ上と組込機器上では、機能は同じであるがプログラム構造が異なるライブラリがリンクされてオブジェクトが生成される。それらオブジェクトは、プログラム構造が異なるので、当然に使用するリソースの量も完全には一致しない。したがって、ＰＣ上では適切に動作した組込用アプリケーションでも、実際に組込機器に組み込んだときに、使用するリソースの量が大きすぎるため、複数の組込用アプリケーションが組込機器上で起動した場合に、他の組込用アプリケーションと競合して動作が不安定となる可能性があった。特に、プリンタ等の組込機器は、ＰＣと異なり使用可能なリソースの量が極端に少ないため、個々の組込用アプリケーションが使用するリソースの量を詳細に管理することは安定動作を実現する上で極めて重要である。
【０００６】
一方、特許文献２記載の発明にあっては、あくまで組込用アプリケーションを組込機器上で実行した場合に動作が不安定になるのを防止するものであり、組込用アプリケーションが使用するリソースの量を組込機器への導入前に検証することはできない。
このような問題は、組込用アプリケーションを組込機器上で実行する場合に限らず、特

定の実行環境で実行させるソフトウェアを他の実行環境で開発するあらゆる場合についても同様に想定される。

【０００７】

そこで、本発明は、このような従来の技術の有する未解決の課題に着目してなされたものであって、ソフトウェアがその実行環境で使用するリソースの量をその実行環境への導入前に検証することにより、ソフトウェアの開発を容易にするとともに安定性の高いソフトウェアを開発するのに好適なリソース管理システムおよびリソース管理プログラム、並びにリソース管理方法を提供することを目的としている。

【課題を解決するための手段】

【０００８】

〔形態１〕　上記目的を達成するために、形態１のリソース管理システムは、

機能モジュールが使用するリソースを管理するリソース管理システムであって、

前記機能モジュールが第１実行環境で使用するリソースの量を測定するリソース測定手段と、第２実行環境でのリソースの上限値を取得するリソース上限値取得手段と、前記リソース上限値取得手段で取得したリソースの上限値を、前記第１実行環境でのリソースの上限値に換算するリソース換算手段と、前記リソース測定手段で測定したリソースの量および前記リソース換算手段で換算したリソースの上限値に基づいてリソースの制限に関する通知を行うリソース制限通知手段とを備えることを特徴とする。

【０００９】

このような構成であれば、リソース測定手段により、機能モジュールが第１実行環境で使用するリソースの量が測定される。また、リソース上限値取得手段により、第２実行環境でのリソースの上限値が取得され、リソース換算手段により、取得されたリソースの上限値が、第１実行環境でのリソースの上限値に換算される。そして、リソース制限通知手段により、測定されたリソースの量および換算されたリソースの上限値に基づいてリソースの制限に関する通知が行われる。

【００１０】

これにより、機能モジュールが第２実行環境で使用するリソースの量が、リソースの上限値に達するかを第２実行環境への導入前に検証することができる。したがって、従来に比して、ソフトウェアの開発を容易に行うことができるとともに安定性の高いソフトウェアを開発することができるという効果が得られる。

ここで、リソースとは、機能モジュールが使用可能な資源をいい、ハードウェア資源に限らず、ソフトウェア資源その他の資源が含まれる。以下、形態２および４のリソース管理システム、形態１３、１４および１６のリソース管理プログラム、並びに形態２５ないし２８、３０および３１のリソース管理方法において同じである。

【００１１】

また、リソースの量としては、例えば、機能モジュールが使用するメモリ量、または起動可能な機能モジュールの数が含まれる。また、例えば、機能モジュールを利用するアプリケーションが確保するリソースの量（メモリ量、機能モジュール数）が含まれる。以下、形態２および４のリソース管理システム、形態１３、１４および１６のリソース管理プログラム、並びに形態２５ないし２８、３０および３１のリソース管理方法において同じである。

【００１２】

また、リソースの換算方法としては、例えば、次の方法が考えられる。第１の方法（一定割合換算方法）は、測定したリソース量に対して一定の割合のリソース量を増加または減少して換算後のリソース量を求める。第２の方法（一定量換算方法）は、測定したリソース量に関わらず、そのリソース量から一定量を増加または減少して換算後のリソース量を求める。第３の方法（定数換算方法）は、測定したリソース量に関わらず、そのリソース量を定数に置き換えて換算後のリソース量を求める。第４の方法（混合型換算方法）は、第１～第３の方法のなかからリソースの種類または測定したリソース量に基づいて選択し、選択した方法により換算後のリソース量を求める。換算方法の選択は、例えば、次の

ように閾値Ａ、Ｂを設定して行うことができる。リソース量＜Ａのときは、定数換算方法を選択し、Ａ＜リソース量＜Ｂのときは、一定量換算方法を選択し、Ｂ＜リソース量のときは、一定割合換算方法を選択する。ただし、Ａ＜Ｂである。以下、形態２および４のリソース管理システム、形態１３、１４および１６のリソース管理プログラム、並びに形態２５ないし２８、３０および３１のリソース管理方法において同じである。

【００１３】

また、リソースを測定することとしては、例えば、アプリケーションがリソースを確保する動作を捕捉し、新たに確保しようとしているリソース量を取得する動作が含まれる。以下、形態２および４のリソース管理システム、形態１３、１４および１６のリソース管理プログラム、並びに形態２５ないし２８、３０および３１のリソース管理方法において同じである。

【００１４】

また、リソースの制限に関する通知としては、例えば、機能モジュールが使用するリソースの量が制限条件若しくは上限値に達したことを示す通知、機能モジュールが使用するリソースの量が制限されたことを示す通知、または機能モジュールが使用するリソースの量を制限すべきことを注意若しくは推奨する通知が含まれる。また、例えば、機能モジュールが使用するリソースの量が上限に達したことを示すメッセージを表示すること、機能モジュールが使用するリソースの量が上限に達したことを示すログ情報を生成することが含まれる。以下、形態２および４のリソース管理システム、形態１３、１４および１６のリソース管理プログラム、並びに形態２５ないし２８、３０および３１のリソース管理方法において同じである。

【００１５】

また、リソースの制限条件としては、例えば、第２実行環境でのリソースの上限値を設定することができる。以下、形態２および４のリソース管理システム、形態１３、１４および１６のリソース管理プログラム、並びに形態２５ないし２８、３０および３１のリソース管理方法において同じである。

また、リソース上限値取得手段は、リソースの上限値を取得するようになっていればどのような構成であってもよく、例えば、入力装置等からリソースの上限値を入力するようになっていてもよいし、外部の装置等からリソースの上限値を獲得または受信するようになっていてもよいし、記憶装置や記憶媒体等からリソースの上限値を読み出すようになっていてもよいし、機能モジュールその他のデータからリソースの上限値を抽出するようになっていてもよい。したがって、取得には、少なくとも入力、獲得、受信、読出および抽出が含まれる。以下、形態２および４のリソース管理システムにおいて同じである。

【００１６】

また、本システムは、単一の装置、端末その他の機器として実現するようにしてもよいし、複数の装置、端末その他の機器を通信可能に接続したネットワークシステムとして実現するようにしてもよい。後者の場合、各構成要素は、それぞれ通信可能に接続されていれば、複数の機器等のうちいずれに属していてもよい。以下、形態２および４のリソース管理システムにおいて同じである。

【００１７】

〔形態２〕　さらに、形態２のリソース管理システムは、

第１機能モジュールと、その実行に前記第１機能モジュールを必要とする複数の第２機能モジュールとがが使用するリソースを管理するリソース管理システムであって、

前記第２機能モジュールが第１実行環境で使用するリソースの量を測定するリソース測定手段と、第２実行環境でのリソースの上限値を取得するリソース上限値取得手段と、前記リソース上限値取得手段で取得したリソースの上限値を、前記第１実行環境でのリソースの上限値に換算するリソース換算手段と、前記リソース測定手段で測定したリソースの量および前記リソース換算手段で換算したリソースの上限値に基づいてリソースの制限に関する通知を行うリソース制限通知手段とを備えることを特徴とする。

【００１８】

このような構成であれば、リソース測定手段により、第２機能モジュールが第１実行環境で使用するリソースの量が測定される。また、リソース上限値取得手段により、第２実行環境でのリソースの上限値が取得され、リソース換算手段により、取得されたリソースの上限値が、第１実行環境でのリソースの上限値に換算される。そして、リソース制限通知手段により、測定されたリソースの量および換算されたリソースの上限値に基づいてリソースの制限に関する通知が行われる。

これにより、第２機能モジュールが第２実行環境で使用するリソースの量が、リソースの上限値に達するかを第２実行環境への導入前に検証することができる。したがって、従来に比して、ソフトウェアの開発を容易に行うことができるとともに安定性の高いソフトウェアを開発することができるという効果が得られる。

【００１９】

〔形態３〕　さらに、形態３のリソース管理システムは、形態２のリソース管理システムにおいて、

前記リソース測定手段は、前記第２機能モジュールが前記第１実行環境で使用するリソースの量、および前記第１機能モジュールが前記第１実行環境で当該第２機能モジュールの実行に使用するリソースの量を測定するようになっていることを特徴とする。

【００２０】

このような構成であれば、リソース測定手段により、第２機能モジュールが第１実行環境で使用するリソースの量、および第１機能モジュールが第１実行環境でその第２機能モジュールの実行に使用するリソースの量が測定される。

これにより、第１機能モジュールが第２実行環境で使用するリソースの量が、リソースの上限値に達するかを第２機能モジュール単位で検証することができるという効果が得られる。

【００２１】

〔形態４〕　さらに、形態４のリソース管理システムは、

第１機能モジュールと、その実行に前記第１機能モジュールを必要とする複数の第２機能モジュールとが使用するリソースを管理するリソース管理システムであって、

前記第１機能モジュールが前記第１実行環境で前記第２機能モジュールの実行に使用するリソースの量を測定するリソース測定手段と、第２実行環境でのリソースの上限値を取得するリソース上限値取得手段と、前記リソース上限値取得手段で取得したリソースの上限値を、前記第１実行環境でのリソースの上限値に換算するリソース換算手段と、前記リソース測定手段で測定したリソースの量および前記リソース換算手段で換算したリソースの上限値に基づいてリソースの制限に関する通知を行うリソース制限通知手段とを備えることを特徴とする。

【００２２】

このような構成であれば、リソース測定手段により、第１機能モジュールが第１実行環境で第２機能モジュールの実行に使用するリソースの量が測定される。また、リソース上限値取得手段により、第２実行環境でのリソースの上限値が取得され、リソース換算手段により、取得されたリソースの上限値が、第１実行環境でのリソースの上限値に換算される。そして、リソース制限通知手段により、測定されたリソースの量および換算されたリソースの上限値に基づいてリソースの制限に関する通知が行われる。

【００２３】

これにより、第１機能モジュールが第２実行環境で第２機能モジュールの実行に使用するリソースの量が、リソースの上限値に達するかを第２実行環境への導入前に検証することができる。したがって、従来に比して、ソフトウェアの開発を容易に行うことができるとともに安定性の高いソフトウェアを開発することができるという効果が得られる。

【００２４】

〔形態５〕　さらに、形態５のリソース管理システムは、形態２ないし４のいずれか１のリソース管理システムにおいて、

前記リソース測定手段で測定したリソースの量および前記リソース換算手段で換算した

リソースの上限値に基づいて前記機能モジュールによるリソースの確保を禁止するリソース制限手段を備えることを特徴とする。

【００２５】

このような構成であれば、リソース制限手段により、測定されたリソースの量および換算されたリソースの上限値に基づいて機能モジュールによるリソースの確保が禁止される。

これにより、機能モジュールが上限を超えてリソースの量を使用するのを制限することができるという効果が得られる。

【００２６】

〔形態６〕　さらに、形態６のリソース管理システムは、形態１ないし５のいずれか１のリソース管理システムにおいて、

前記リソース換算手段は、前記第１実行環境で所定条件で使用されるリソースの量および前記第２実行環境で前記所定条件で使用されるリソースの量に基づき決定される換算率を登録したリソース換算テーブルに基づいて換算を行うようになっていることを特徴とする。

【００２７】

このような構成であれば、リソース換算手段により、リソース換算テーブルに基づいて換算が行われる。

これにより、第１実行環境および第２実行環境の間でリソースの量または上限値を比較的正確に換算することができるという効果が得られる。

ここで、所定条件としては、例えば、第１実行環境、第２実行環境どちらでも実行可能なテストプログラムを同一の環境下（リソースという意味でなく、他のプログラムとの同時実行等）で実行することが含まれる。以下、形態１８のリソース管理プログラム、および形態３３のリソース管理方法において同じである。

【００２８】

また、リソースの量に基づき決定される換算率とは、例えば、テストプログラムを実行した結果得られる２つの実行環境間でのリソース消費を比較して算出した比率をいう。すなわち、決定には算出が含まれる。以下、形態１８のリソース管理プログラム、および形態３３のリソース管理方法において同じである。

【００２９】

〔形態７〕　さらに、形態７のリソース管理システムは、形態６のリソース管理システムにおいて、

前記リソース換算テーブルは、リソースの種別ごとに前記換算率を登録したものであり、

前記リソース換算手段は、前記機能モジュールが使用するリソースの種別に応じて前記リソース換算テーブルから対応する前記換算率を取得し、取得した換算率に基づいて換算を行うようになっていることを特徴とする。

【００３０】

このような構成であれば、リソース換算手段により、機能モジュールが使用するリソースの種別に応じてリソース換算テーブルから対応する換算率が取得され、取得された換算率に基づいて換算が行われる。

これにより、機能モジュールが使用するリソースの種別に応じた換算を行うことができるので、第１実行環境および第２実行環境の間でリソースの量または上限値をさらに正確に換算することができるという効果が得られる。

【００３１】

ここで、リソースの種別としては、例えば、メモリ、ファイルシステム等のハードウェアの占有割合で定義できるリソースの場合には、そのハードウェアの種別が、スレッドやソケット等のＯＳ（Operating System）に依存するソフトウェア的な概念の場合にはその概念の種別がそれぞれ含まれる。以下、形態１９のリソース管理プログラム、および形態３４のリソース管理方法において同じである。

【００３２】
また、リソース換算テーブルは、リソースの種別ごとに換算率を登録したものであればどのような構成であってもよいが、さらに具体的には、リソースの種別に基づいて換算方法を選択し、選択した換算方法によりリソースの種別ごとに換算率を算出し、算出した換算率を登録することもできる。換算方法としては、例えば、上記のように、一定割合換算方法、一定量換算方法、定数換算方法、混合型換算方法が考えられる。以下、形態１９のリソース管理プログラム、および形態３４のリソース管理方法において同じである。
【００３３】
〔形態８〕　さらに、形態８のリソース管理システムは、形態６および７のいずれか１のリソース管理システムにおいて、
前記リソース換算テーブルは、リソースの使用形態ごとに前記換算率を登録したものであり、
前記リソース換算手段は、前記機能モジュールが使用するリソースの形態に応じて前記リソース換算テーブルから対応する前記換算率を取得し、取得した換算率に基づいて換算を行うようになっていることを特徴とする。
【００３４】
このような構成であれば、リソース換算手段により、機能モジュールが使用するリソースの形態に応じてリソース換算テーブルから対応する換算率が取得され、取得された換算率に基づいて換算が行われる。
これにより、機能モジュールが使用するリソースの形態に応じた換算を行うことができるので、第１実行環境および第２実行環境の間でリソースの量または上限値をさらに正確に換算することができるという効果が得られる。
【００３５】
ここで、リソースの使用形態とは、同じ種別のリソースのなかで使用目的が異なるものをいう。具体的には、メモリの例でいえば、ＪＡＶＡ（登録商標）インタプリタで確保されるメモリやＺｉｐライブラリで確保されるメモリというふうに、使用形態が異なる場合があり、それぞれ換算率が異なっていますので、別々に取り扱う必要がある。以下、形態２０のリソース管理プログラム、および形態３５のリソース管理方法において同じである。
【００３６】
〔形態９〕　さらに、形態９のリソース管理システムは、形態６ないし８のいずれか１のリソース管理システムにおいて、
前記リソース換算テーブルは、複数のテストモジュールのそれぞれについて、前記第１実行環境および前記第２実行環境で当該テストモジュールが使用するリソースの量に基づいて前記換算率が決定され、前記各テストモジュールについて決定された前記換算率のうち最大値を登録したものであることを特徴とする。
【００３７】
これにより、リソース換算テーブルには、各テストモジュールについて決定された換算率のうち最大値が登録されているので、機能モジュールが使用するリソースの量を多めに見積もることができる。したがって、機能モジュールの動作を比較的確実に保証することができるという効果が得られる。
ここで、テストモジュールとは、第１実行環境および第２実行環境の両方で動作し、その実行によりリソースを消費するものをいう。以下、形態１０のリソース管理システム、形態２１および２２のリソース管理プログラム、並びに形態３６および３７のリソース管理方法において同じである。
【００３８】
〔形態１０〕　さらに、形態１０のリソース管理システムは、形態６ないし８のいずれか１のリソース管理システムにおいて、
前記リソース換算テーブルは、複数のテストモジュールのそれぞれについて、前記第１実行環境および前記第２実行環境で当該テストモジュールが使用するリソースの量に基づ

いて前記換算率が決定され、前記各テストモジュールについて決定された前記換算率の平均値を登録したものであることを特徴とする。

【００３９】

これにより、リソース換算テーブルには、各テストモジュールについて決定された換算率のうち平均値が登録されているので、機能モジュールの動作をある程度の確実性をもって保証することができるとともに第２実行環境で使用されるリソースの量を抑制することができるという効果が得られる。

【００４０】

〔形態１１〕　さらに、形態１１のリソース管理システムは、形態１ないし１０のいずれか１のリソース管理システムにおいて、

前記リソース制限通知手段は、前記機能モジュールが使用するリソースの量が上限に達したことを示すメッセージを表示するようになっていることを特徴とする。

【００４１】

このような構成であれば、リソース制限通知手段により、機能モジュールが使用するリソースの量が上限に達したことを示すメッセージが表示される。

これにより、メッセージを参照することにより機能モジュールが使用するリソースの量が上限に達したことを把握することができるという効果が得られる。

ここで、リソース制限通知手段は、メッセージを表示するようになっているが、他の形態としては、音声、画面のＦｌｕｓｈ、アイコンの表示、メール、電話、ＦＡＸ等により通知を行うことが考えられる。以下、形態２３のリソース管理プログラム、および形態３８のリソース管理方法において同じである。

【００４２】

〔形態１２〕　さらに、形態１２のリソース管理システムは、形態１ないし１０のいずれか１のリソース管理システムにおいて、

前記リソース制限通知手段は、前記機能モジュールが使用するリソースの量が上限に達したことを示すログ情報を生成するようになっていることを特徴とする。

このような構成であれば、リソース制限通知手段により、機能モジュールが使用するリソースの量が上限に達したことを示すログ情報が生成される。

これにより、ログ情報を参照することにより機能モジュールが使用するリソースの量が上限に達したことを把握することができるという効果が得られる。

【００４３】

〔形態１３〕　一方、上記目的を達成するために、形態１３のリソース管理プログラムは、

機能モジュールが使用するリソースを管理するリソース管理プログラムであって、

前記機能モジュールが第１実行環境で使用するリソースの量を測定するリソース測定ステップと、第２実行環境でのリソースの上限値を取得するリソース上限値取得ステップと、前記リソース上限値取得ステップで取得したリソースの上限値を、前記第１実行環境でのリソースの上限値に換算するリソース換算ステップと、前記リソース測定ステップで測定したリソースの量および前記リソース換算ステップで換算したリソースの上限値に基づいてリソースの制限に関する通知を行うリソース制限通知ステップとからなる処理をコンピュータに実行させるためのプログラムを含むことを特徴とする。

【００４４】

このような構成であれば、コンピュータによってプログラムが読み取られ、読み取られたプログラムに従ってコンピュータが処理を実行すると、形態１のリソース管理システムと同等の作用および効果が得られる。

ここで、リソース上限値取得ステップは、リソースの上限値を取得すればどのような形態であってもよく、例えば、入力装置等からリソースの上限値を入力してもよいし、外部の装置等からリソースの上限値を獲得または受信してもよいし、記憶装置や記憶媒体等からリソースの上限値を読み出してもよいし、機能モジュールその他のデータからリソースの上限値を抽出してもよい。したがって、取得には、少なくとも入力、獲得、受信、読出

および抽出が含まれる。以下、形態１４および１６のリソース管理プログラム、並びに形態２５ないし２８、３０および３１のリソース管理方法において同じである。

【００４５】

〔形態１４〕　さらに、形態１４のリソース管理プログラムは、

第１機能モジュールと、その実行に前記第１機能モジュールを必要とする複数の第２機能モジュールとが使用するリソースを管理するリソース管理プログラムであって、

前記第２機能モジュールが第１実行環境で使用するリソースの量を測定するリソース測定ステップと、第２実行環境でのリソースの上限値を取得するリソース上限値取得ステップと、前記リソース上限値取得ステップで取得したリソースの上限値を、前記第１実行環境でのリソースの上限値に換算するリソース換算ステップと、前記リソース測定ステップで測定したリソースの量および前記リソース換算ステップで換算したリソースの上限値に基づいてリソースの制限に関する通知を行うリソース制限通知ステップとからなる処理をコンピュータに実行させるためのプログラムを含むことを特徴とする。

【００４６】

このような構成であれば、コンピュータによってプログラムが読み取られ、読み取られたプログラムに従ってコンピュータが処理を実行すると、形態２のリソース管理システムと同等の作用および効果が得られる。

【００４７】

〔形態１５〕　さらに、形態１５のリソース管理プログラムは、形態１４のリソース管理プログラムにおいて、

前記リソース測定ステップは、前記第２機能モジュールが前記第１実行環境で使用するリソースの量、および前記第１機能モジュールが前記第１実行環境で当該第２機能モジュールの実行に使用するリソースの量を測定することを特徴とする。

このような構成であれば、コンピュータによってプログラムが読み取られ、読み取られたプログラムに従ってコンピュータが処理を実行すると、形態３のリソース管理システムと同等の作用および効果が得られる。

【００４８】

〔形態１６〕　さらに、形態１６のリソース管理プログラムは、

第１機能モジュールと、その実行に前記第１機能モジュールを必要とする複数の第２機能モジュールとが使用するリソースを管理するリソース管理プログラムであって、

前記第１機能モジュールが前記第１実行環境で前記第２機能モジュールの実行に使用するリソースの量を測定するリソース測定ステップと、第２実行環境でのリソースの上限値を取得するリソース上限値取得ステップと、前記リソース上限値取得ステップで取得したリソースの上限値を、前記第１実行環境でのリソースの上限値に換算するリソース換算ステップと、前記リソース測定ステップで測定したリソースの量および前記リソース換算ステップで換算したリソースの上限値に基づいてリソースの制限に関する通知を行うリソース制限通知ステップとからなる処理をコンピュータに実行させるためのプログラムを含むことを特徴とする。

このような構成であれば、コンピュータによってプログラムが読み取られ、読み取られたプログラムに従ってコンピュータが処理を実行すると、形態４のリソース管理システムと同等の作用および効果が得られる。

【００４９】

〔形態１７〕　さらに、形態１７のリソース管理プログラムは、形態１４ないし１６のいずれか１のリソース管理プログラムにおいて、

前記リソース測定ステップで測定したリソースの量および前記リソース換算ステップで換算したリソースの上限値に基づいて前記機能モジュールによるリソースの確保を禁止するリソース制限ステップからなる処理をコンピュータに実行させるためのプログラムを含むことを特徴とする。

このような構成であれば、コンピュータによってプログラムが読み取られ、読み取られたプログラムに従ってコンピュータが処理を実行すると、形態５のリソース管理システム

と同等の作用および効果が得られる。

【００５０】

〔形態１８〕　さらに、形態１８のリソース管理プログラムは、形態１３ないし１７のいずれか１のリソース管理プログラムにおいて、

前記リソース換算ステップは、前記第１実行環境で所定条件で使用されるリソースの量および前記第２実行環境で前記所定条件で使用されるリソースの量に基づき決定される換算率を登録したリソース換算テーブルに基づいて換算を行うことを特徴とする。

このような構成であれば、コンピュータによってプログラムが読み取られ、読み取られたプログラムに従ってコンピュータが処理を実行すると、形態６のリソース管理システムと同等の作用および効果が得られる。

【００５１】

〔形態１９〕　さらに、形態１９のリソース管理プログラムは、形態１８のリソース管理プログラムにおいて、

前記リソース換算テーブルは、リソースの種別ごとに前記換算率を登録したものであり、

前記リソース換算ステップは、前記機能モジュールが使用するリソースの種別に応じて前記リソース換算テーブルから対応する前記換算率を取得し、取得した換算率に基づいて換算を行うことを特徴とする。

このような構成であれば、コンピュータによってプログラムが読み取られ、読み取られたプログラムに従ってコンピュータが処理を実行すると、形態７のリソース管理システムと同等の作用および効果が得られる。

【００５２】

〔形態２０〕　さらに、形態２０のリソース管理プログラムは、形態１８および１９のいずれか１のリソース管理プログラムにおいて、

前記リソース換算テーブルは、リソースの使用形態ごとに前記換算率を登録したものであり、

前記リソース換算ステップは、前記機能モジュールが使用するリソースの形態に応じて前記リソース換算テーブルから対応する前記換算率を取得し、取得した換算率に基づいて換算を行うことを特徴とする。

このような構成であれば、コンピュータによってプログラムが読み取られ、読み取られたプログラムに従ってコンピュータが処理を実行すると、形態８のリソース管理システムと同等の作用および効果が得られる。

【００５３】

〔形態２１〕　さらに、形態２１のリソース管理プログラムは、形態１８ないし２０のいずれか１のリソース管理プログラムにおいて、

前記リソース換算テーブルは、複数のテストモジュールのそれぞれについて、前記第１実行環境および前記第２実行環境で当該テストモジュールが使用するリソースの量に基づいて前記換算率が決定され、前記各テストモジュールについて決定された前記換算率のうち最大値を登録したものであることを特徴とする。

このような構成であれば、コンピュータによってプログラムが読み取られ、読み取られたプログラムに従ってコンピュータが処理を実行すると、形態９のリソース管理システムと同等の作用および効果が得られる。

【００５４】

〔形態２２〕　さらに、形態２２のリソース管理プログラムは、形態１８ないし２０のいずれか１のリソース管理プログラムにおいて、

前記リソース換算テーブルは、複数のテストモジュールのそれぞれについて、前記第１実行環境および前記第２実行環境で当該テストモジュールが使用するリソースの量に基づいて前記換算率が決定され、前記各テストモジュールについて決定された前記換算率の平均値を登録したものであることを特徴とする。

このような構成であれば、コンピュータによってプログラムが読み取られ、読み取られ

たプログラムに従ってコンピュータが処理を実行すると、形態１０のリソース管理システムと同等の作用および効果が得られる。

【００５５】

〔形態２３〕　さらに、形態２３のリソース管理プログラムは、形態１３ないし２２のいずれか１のリソース管理プログラムにおいて、

前記リソース制限通知ステップは、前記機能モジュールが使用するリソースの量が上限に達したことを示すメッセージを表示することを特徴とする。

このような構成であれば、コンピュータによってプログラムが読み取られ、読み取られたプログラムに従ってコンピュータが処理を実行すると、形態１１のリソース管理システムと同等の作用および効果が得られる。

【００５６】

〔形態２４〕　さらに、形態２４のリソース管理プログラムは、形態１３ないし２２のいずれか１のリソース管理プログラムにおいて、

前記リソース制限通知ステップは、前記機能モジュールが使用するリソースの量が上限に達したことを示すログ情報を生成することを特徴とする。

このような構成であれば、コンピュータによってプログラムが読み取られ、読み取られたプログラムに従ってコンピュータが処理を実行すると、形態１２のリソース管理システムと同等の作用および効果が得られる。

【００５７】

〔形態２５〕　一方、上記目的を達成するために、形態２５のリソース管理方法は、

機能モジュールが使用するリソースを管理するリソース管理方法であって、

前記機能モジュールが第１実行環境で使用するリソースの量を測定するリソース測定ステップと、第２実行環境でのリソースの上限値を取得するリソース上限値取得ステップと、前記リソース上限値取得ステップで取得したリソースの上限値を、前記第１実行環境でのリソースの上限値に換算するリソース換算ステップと、前記リソース測定ステップで測定したリソースの量および前記リソース換算ステップで換算したリソースの上限値に基づいてリソースの制限に関する通知を行うリソース制限通知ステップとを含むことを特徴とする。

これにより、形態１のリソース管理システムと同等の効果が得られる。

【００５８】

〔形態２６〕　さらに、形態２６のリソース管理方法は、

機能モジュールが使用するリソースを管理するリソース管理方法であって、

演算手段が、前記機能モジュールが第１実行環境で使用するリソースの量を測定するリソース測定ステップと、

前記演算手段が、第２実行環境でのリソースの上限値を取得するリソース上限値取得ステップと、

前記演算手段が、前記リソース上限値取得ステップで取得したリソースの上限値を、前記第１実行環境でのリソースの上限値に換算するリソース換算ステップと、

前記演算手段が、前記リソース測定ステップで測定したリソースの量および前記リソース換算ステップで換算したリソースの上限値に基づいてリソースの制限に関する通知を行うリソース制限通知ステップとを含むことを特徴とする。

これにより、形態１のリソース管理システムと同等の効果が得られる。

【００５９】

〔形態２７〕　さらに、形態２７のリソース管理方法は、

第１機能モジュールと、その実行に前記第１機能モジュールを必要とする複数の第２機能モジュールとが使用するリソースを管理するリソース管理方法であって、

前記第２機能モジュールが第１実行環境で使用するリソースの量を測定するリソース測定ステップと、第２実行環境でのリソースの上限値を取得するリソース上限値取得ステップと、前記リソース上限値取得ステップで取得したリソースの上限値を、前記第１実行環境でのリソースの上限値に換算するリソース換算ステップと、前記リソース測定ステップ

で測定したリソースの量および前記リソース換算ステップで換算したリソースの上限値に基づいてリソースの制限に関する通知を行うリソース制限通知ステップとを含むことを特徴とする。

これにより、形態２のリソース管理システムと同等の効果が得られる。

【００６０】

〔形態２８〕　さらに、形態２８のリソース管理方法は、

第１機能モジュールと、その実行に前記第１機能モジュールを必要とする複数の第２機能モジュールとが使用するリソースを管理するリソース管理方法であって、

演算手段が、前記第２機能モジュールが第１実行環境で使用するリソースの量を測定するリソース測定ステップと、

前記演算手段が、第２実行環境でのリソースの上限値を取得するリソース上限値取得ステップと、

前記演算手段が、前記リソース上限値取得ステップで取得したリソースの上限値を、前記第１実行環境でのリソースの上限値に換算するリソース換算ステップと、

前記演算手段が、前記リソース測定ステップで測定したリソースの量および前記リソース換算ステップで換算したリソースの上限値に基づいてリソースの制限に関する通知を行うリソース制限通知ステップとを含むことを特徴とする。

これにより、形態２のリソース管理システムと同等の効果が得られる。

【００６１】

〔形態２９〕　さらに、形態２９のリソース管理方法は、形態２７および２８のいずれか１のリソース管理方法において、

前記リソース測定ステップは、前記第２機能モジュールが前記第１実行環境で使用するリソースの量、および前記第１機能モジュールが前記第１実行環境で当該第２機能モジュールの実行に使用するリソースの量を測定することを特徴とする。

これにより、形態３のリソース管理システムと同等の効果が得られる。

【００６２】

〔形態３０〕　さらに、形態３０のリソース管理方法は、

第１機能モジュールと、その実行に前記第１機能モジュールを必要とする複数の第２機能モジュールとが使用するリソースを管理するリソース管理方法であって、

前記第１機能モジュールが前記第１実行環境で前記第２機能モジュールの実行に使用するリソースの量を測定するリソース測定ステップと、第２実行環境でのリソースの上限値を取得するリソース上限値取得ステップと、前記リソース上限値取得ステップで取得したリソースの上限値を、前記第１実行環境でのリソースの上限値に換算するリソース換算ステップと、前記リソース測定ステップで測定したリソースの量および前記リソース換算ステップで換算したリソースの上限値に基づいてリソースの制限に関する通知を行うリソース制限通知ステップとを含むことを特徴とする。

これにより、形態４のリソース管理システムと同等の効果が得られる。

【００６３】

〔形態３１〕　さらに、形態３１のリソース管理方法は、

第１機能モジュールと、その実行に前記第１機能モジュールを必要とする複数の第２機能モジュールとが使用するリソースを管理するリソース管理方法であって、

演算手段が、前記第１機能モジュールが前記第１実行環境で前記第２機能モジュールの実行に使用するリソースの量を測定するリソース測定ステップと、

前記演算手段が、第２実行環境でのリソースの上限値を取得するリソース上限値取得ステップと、

前記演算手段が、前記リソース上限値取得ステップで取得したリソースの上限値を、前記第１実行環境でのリソースの上限値に換算するリソース換算ステップと、

前記演算手段が、前記リソース測定ステップで測定したリソースの量および前記リソース換算ステップで換算したリソースの上限値に基づいてリソースの制限に関する通知を行うリソース制限通知ステップとを含むことを特徴とする。

これにより、形態４のリソース管理システムと同等の効果が得られる。

【００６４】

〔形態３２〕　さらに、形態３２のリソース管理方法は、形態２７ないし３１のいずれか１のリソース管理方法において、

前記リソース測定ステップで測定したリソースの量および前記リソース換算ステップで換算したリソースの上限値に基づいて前記機能モジュールによるリソースの確保を禁止するリソース制限ステップを含むことを特徴とする。

これにより、形態５のリソース管理システムと同等の効果が得られる。

【００６５】

〔形態３３〕　さらに、形態３３のリソース管理方法は、形態２５ないし３２のいずれか１のリソース管理方法において、

前記リソース換算ステップは、前記第１実行環境で所定条件で使用されるリソースの量および前記第２実行環境で前記所定条件で使用されるリソースの量に基づき決定される換算率を登録したリソース換算テーブルに基づいて換算を行うことを特徴とする。

これにより、形態６のリソース管理システムと同等の効果が得られる。

【００６６】

〔形態３４〕　さらに、形態３４のリソース管理方法は、形態３３のリソース管理方法において、

前記リソース換算テーブルは、リソースの種別ごとに前記換算率を登録したものであり、

前記リソース換算ステップは、前記機能モジュールが使用するリソースの種別に応じて前記リソース換算テーブルから対応する前記換算率を取得し、取得した換算率に基づいて換算を行うことを特徴とする。

これにより、形態７のリソース管理システムと同等の効果が得られる。

【００６７】

〔形態３５〕　さらに、形態３５のリソース管理方法は、形態３３および３４のいずれか１のリソース管理方法において、

前記リソース換算テーブルは、リソースの使用形態ごとに前記換算率を登録したものであり、

前記リソース換算ステップは、前記機能モジュールが使用するリソースの形態に応じて前記リソース換算テーブルから対応する前記換算率を取得し、取得した換算率に基づいて換算を行うことを特徴とする。

これにより、形態８のリソース管理システムと同等の効果が得られる。

【００６８】

〔形態３６〕　さらに、形態３６のリソース管理方法は、形態３３ないし３５のいずれか１のリソース管理方法において、

前記リソース換算テーブルは、複数のテストモジュールのそれぞれについて、前記第１実行環境および前記第２実行環境で当該テストモジュールが使用するリソースの量に基づいて前記換算率が決定され、前記各テストモジュールについて決定された前記換算率のうち最大値を登録したものであることを特徴とする。

これにより、形態９のリソース管理システムと同等の効果が得られる。

【００６９】

〔形態３７〕　さらに、形態３７のリソース管理方法は、形態３３ないし３５のいずれか１のリソース管理方法において、

前記リソース換算テーブルは、複数のテストモジュールのそれぞれについて、前記第１実行環境および前記第２実行環境で当該テストモジュールが使用するリソースの量に基づいて前記換算率が決定され、前記各テストモジュールについて決定された前記換算率の平均値を登録したものであることを特徴とする。

これにより、形態１０のリソース管理システムと同等の効果が得られる。

【００７０】

〔形態３８〕　さらに、形態３８のリソース管理方法は、形態２５ないし３７のいずれか１のリソース管理方法において、
前記リソース制限通知ステップは、前記機能モジュールが使用するリソースの量が上限に達したことを示すメッセージを表示することを特徴とする。
これにより、形態１１のリソース管理システムと同等の効果が得られる。
【００７１】
〔形態３９〕　さらに、形態３９のリソース管理方法は、形態２５ないし３７のいずれか１のリソース管理方法において、
前記リソース制限通知ステップは、前記機能モジュールが使用するリソースの量が上限に達したことを示すログ情報を生成することを特徴とする。
これにより、形態１２のリソース管理システムと同等の効果が得られる。
【発明を実施するための最良の形態】
【００７２】
以下、本発明の第１の実施の形態を図面を参照しながら説明する。図１ないし図２０は、本発明に係るリソース管理システムおよび、並びにの第１の実施の形態を示す図である。
本実施の形態は、本発明に係るリソース管理システムおよび、並びにを、図２に示すように、ホスト端末１００上のＪＡＶＡ（登録商標）アプリケーションの実行環境において、ネットワークプリンタの動作を制御するためのＪＡＶＡ（登録商標）クラスセットをエミュレーションする場合について適用したものである。
【００７３】
まず、本発明を適用するホスト端末１００の機能概要を説明する。
図１は、ＪＡＶＡ（登録商標）ソフトウェアの構成を示す図である。
ＪＡＶＡ（登録商標）アプリケーションの実行環境では、ＪＡＶＡ（登録商標）クラスセットの実行を制御するＪＡＶＡ（登録商標）クラスおよびＪＶＭ（Java（登録商標） Virtual Machine）からなる共通機能モジュールをＯＳ上で実行し、ＪＡＶＡ（登録商標）クラスセットである個別機能モジュールを共通機能モジュール上で実行する。ここで、ＪＡＶＡ（登録商標）ソフトウェアは、共通機能モジュールおよび個別機能モジュールから構成される。
【００７４】
共通機能モジュールは、図１に示すように、複数の個別機能モジュールを実行可能となっている。図１の例では、共通機能モジュールａ上で２つの個別機能モジュールｂ，ｃが実行されている場合を示している。ここで、個別機能モジュールｂが使用するリソースの量をｘ１、共通機能モジュールａが個別機能モジュールｂの実行に使用するリソースの量をｘ２、個別機能モジュールｂが使用可能なリソースの上限値をX_{max}とした場合、本実施の形態では、$x_1 + x_2 \leqq X_{max}$となるようにリソースの量を制限する。
【００７５】
図２は、ホスト端末１００の機能概要を示す機能ブロック図である。
ホスト端末１００は、図２に示すように、ＯＳ１１０と、共通機能モジュール１２０と、複数の個別機能モジュール１３０とを有して構成されている。
ＯＳ１１０は、ＪＡＶＡ（登録商標）ソフトウェアが使用するリソースの量を測定するリソース測定部１０と、ＪＡＶＡ（登録商標）ソフトウェア全体が使用するリソースの量を制限するリソース制限部１２とを有して構成されている。
【００７６】
リソース制限部１２は、リソース測定部１０で測定したリソースの量が、ＪＡＶＡ（登録商標）ソフトウェアに割り当てられた所定の上限値未満となるように、ＪＡＶＡ（登録商標）ソフトウェアが使用するリソースを制限する。
共通機能モジュール１２０は、個別機能モジュール１３０の実行を管理する個別機能モジュール管理部１４と、個別機能モジュール管理部１４および個別機能モジュール１３０が使用するリソースの量を測定するリソース測定部１６と、ホスト端末１００で所定条件

で使用されるリソースの量およびネットワークプリンタで同所定条件で使用されるリソースの量に基づき決定される換算率を登録したリソース換算テーブル２２と、リソースの量を換算するリソース換算部２４とを有して構成されている。

【００７７】

リソース測定部１６は、各個別機能モジュール１３０ごとに、その個別機能モジュール１３０が使用するリソースの量、および個別機能モジュール管理部１４がその個別機能モジュール１３０の実行に使用するリソースの量を測定する。

リソース換算部２４は、リソース換算テーブル２２に基づいて、リソース測定部１６で測定したリソースの量を、ネットワークプリンタで使用するリソースの量に換算する。

【００７８】

共通機能モジュール１２０は、さらに、個別機能モジュール１３０がネットワークプリンタでのリソースの上限値を取得する上限値取得部１８と、個別機能モジュール管理部１４および個別機能モジュール１３０が使用するリソースの量を制限するリソース制限部２０と、ログ情報を生成するログ情報生成部２６とを有して構成されている。

リソース制限部２０は、リソース換算部２４で換算したリソースの量が、上限値取得部１８で取得した上限値未満となるように、個別機能モジュール１３０が使用するリソースの量、および個別機能モジュール管理部１４がその個別機能モジュール１３０の実行に使用するリソースの量を制限する。

【００７９】

ログ情報生成部２６は、リソース換算部２４で換算したリソースの量が、上限値取得部１８で取得した上限値以上であると判定したときは、個別機能モジュール１３０が使用するリソースの量が上限に達したことを示すログ情報を生成する。

次に、ホスト端末１００の構成を詳細に説明する。

図３は、ホスト端末１００のハードウェア構成を示すブロック図である。

【００８０】

ホスト端末１００は、図３に示すように、制御プログラムに基づいて演算およびシステム全体を制御するＣＰＵ３０と、所定領域にあらかじめＣＰＵ３０の制御プログラム等を格納しているＲＯＭ３２と、ＲＯＭ３２等から読み出したデータやＣＰＵ３０の演算過程で必要な演算結果を格納するためのＲＡＭ３４と、外部装置に対してデータの入出力を媒介するＩ／Ｆ３８とで構成されており、これらは、データを転送するための信号線であるバス３９で相互にかつデータ授受可能に接続されている。

【００８１】

Ｉ／Ｆ３８には、外部装置として、ヒューマンインターフェースとしてデータの入力が可能なキーボードやマウス等からなる入力装置４０と、データやテーブル等をファイルとして格納する記憶装置４２と、画像信号に基づいて画面を表示する表示装置４４と、ネットワーク１９９に接続するための信号線とが接続されている。

次に、記憶装置４２のデータ構造を詳細に説明する。

【００８２】

記憶装置４２は、共通機能モジュール１２０および複数の個別機能モジュール１３０を記憶している。

個別機能モジュール１３０は、ネットワークプリンタでのリソースの上限値を格納したリソース制限情報を含んで構成されている。

図４は、リソース制限情報４００のデータ構造を示す図である。

【００８３】

リソース制限情報４００は、図４に示すように、個別機能モジュール１３０および共通機能モジュール１２０がその個別機能モジュール１３０の実行に使用可能なメモリ（ＲＡＭ３４）の上限値を格納したフィールド４０２と、個別機能モジュール１３０および共通機能モジュール１２０がその個別機能モジュール１３０の実行に起動可能なクラス数を格納したフィールド４０４とを含んで構成されている。

【００８４】

個別機能モジュール１３０は、さらに、個別機能モジュール１３０に関するモジュール情報を含んで構成されている。

図５は、モジュール情報４２０のデータ構造を示す図である。

モジュール情報４２０は、図５に示すように、個別機能モジュール１３０が使用するリソースの量を制限する対象（以下、リソース管理対象という。）とするか否かを格納したフィールド４２２と、個別機能モジュール１３０が実行可能なネットワークプリンタの機種を格納したフィールド４２４と、電子署名情報を格納したフィールド４２６とを含んで構成されている。

【００８５】

図５の例では、フィールド４２２には「有効」が格納されている。これは、個別機能モジュール１３０をリソース管理対象として管理することを示している。さらに、フィールド４２４には「TypeA」が、フィールド４２６には「Ｘ社」がそれぞれ格納されている。これは、個別機能モジュール１３０が実行可能な機種が「TypeA」であり、Ｘ社の電子署名を受けていることを示している。

【００８６】

記憶装置４２は、さらに、共通機能モジュール１２０の実行環境を示す実行環境情報を登録した実行環境情報登録テーブルを記憶している。

図６は、実行環境情報登録テーブル４４０のデータ構造を示す図である。

実行環境情報登録テーブル４４０は、図６に示すように、個別機能モジュール１３０の起動可能な個数の上限値を登録したフィールド４４２と、実行すべき個別機能モジュール１３０の名称を登録したフィールド４４４と、削除すべき個別機能モジュール１３０の名称を登録したフィールド４４６と、エミュレーションするネットワークプリンタの機種を登録したフィールド４４８と、対応可能な電子署名情報を登録したフィールド４５０とを含んで構成されている。

【００８７】

図６の例では、フィールド４４２には「５」が、フィールド４４４には「個別機能モジュールｂ，ｄ」が、フィールド４４６には「個別機能モジュールｃ」がそれぞれ登録されている。これは、個別機能モジュール１３０を最大５個まで起動することができ、共通機能モジュール１２０の起動時に、個別機能モジュールｂ，ｄを実行し、個別機能モジュールｃを削除すべきことを示している。さらに、フィールド４４８には「TypeA」が、フィールド４５０には「Ｘ社」がそれぞれ登録されている。これは、エミュレーションするネットワークプリンタの機種が「TypeA」であり、Ｘ社の電子署名情報を含む個別機能モジュール１３０を実行できることを示している。

【００８８】

記憶装置４２は、さらに、リソース換算テーブル２２を記憶している。

図７は、リソース換算テーブル２２のデータ構造を示す図である。

リソース換算テーブル２２には、図７に示すように、リソースの種別または使用形態ごとに１つのレコードが登録されている。各レコードは、リソースの名称を登録するフィールド５０２と、換算率を登録するフィールド５０４とを含んで構成されている。

【００８９】

図７の例では、第１段目のレコードには、リソースの名称として「メモリ　使用形態Ａ」が、換算率として「１」がそれぞれ登録されている。これは、個別機能モジュール１３０がメモリを使用形態Ａで使用する場合は、個別機能モジュール１３０がホスト端末１００で使用するメモリ量に換算率「１」を乗算することにより、ネットワークプリンタで使用するメモリ量への換算を行うことを示している。同様に、個別機能モジュール１３０がメモリを使用形態Ｂ，Ｃで使用する場合は、使用形態Ｂ，Ｃに対応する換算率を採用する。

【００９０】

メモリの使用形態Ａ～Ｃは、共通機能モジュール１２０および個別機能モジュール１３０が使用する関数やライブラリによって決定される。例えば、整数型の変数を扱うライブ

ラリ等を使用する場合は使用形態Ａとなり、ダブル型の変数を扱うライブラリ等を使用する場合は使用形態Ｂとなる。

また、第４段目のレコードには、リソースの名称として「クラス数」が、換算率として「１」がそれぞれ登録されている。これは、個別機能モジュール１３０がホスト端末１００で起動するクラス数に換算率「１」を乗算することにより、ネットワークプリンタで起動するクラス数への換算を行うことを示している。

【００９１】

なお、リソース換算テーブル２２には、複数のテストモジュールのそれぞれについて、ホスト端末１００およびネットワークプリンタでそのテストモジュールが使用するリソースの量に基づいて換算率を決定し、各テストモジュールについて決定された換算率のうち最大値が登録されている。リソース換算テーブル２２の生成については、後段の実施の形態で詳述する。

【００９２】

記憶装置４２は、さらに、リソース管理対象となる各個別機能モジュール１３０ごとに、その個別機能モジュール１３０が使用するリソースの量を管理するリソース管理テーブルを記憶している。リソース管理テーブルは、個別機能モジュール１３０がリソース管理対象である場合にその起動に伴って生成される。

図８は、リソース管理テーブル４６０のデータ構造を示す図である。

【００９３】

リソース管理テーブル４６０には、図８に示すように、リソースの種別ごとに１つのレコードが登録されている。各レコードは、リソースの名称を登録するフィールド４６２と、個別機能モジュール１３０がネットワークプリンタでのリソースの上限値を登録するフィールド４６４と、個別機能モジュール１３０がホスト端末１００で現在使用しているリソースの量を登録するフィールド４６６と、フィールド４６６の値を、ネットワークプリンタで使用するリソースの量に換算した値を登録するフィールド４６８とを含んで登録されている。

【００９４】

図８の例では、第１段目のレコードには、リソースの名称として「メモリ」が、上限値として「1000000」が、現在値としてメモリの使用形態Ａ～Ｃごとに「200000」、「100000」および「150000」が、換算値として「650000」がそれぞれ登録されている。これは、個別機能モジュール１３０がネットワークプリンタでのメモリ量の上限値が1000000[byte]であり、現在メモリを、使用形態Ａで200000[byte]、使用形態Ｂで100000[byte]、使用形態Ｃで150000[byte]使用していることを示している。さらに、ネットワークプリンタで使用するメモリ量に換算した値（以下、換算メモリ量という。）が650000[byte]であることを示している。換算メモリ量は、図７のリソース換算テーブル２２を参照し、200000×１＋100000×1.5＋150000×２＝650000として算出することができる。

【００９５】

また、第２段目のレコードには、リソースの名称として「クラス数」が、上限値として「100」が、現在値として「20」が、換算値として「20」がそれぞれ登録されている。これは、個別機能モジュール１３０がネットワークプリンタで起動可能なクラス数の上限値が100個であり、現在クラスを20個起動していることを示している。さらに、ネットワークプリンタで起動するクラス数に換算した値（以下、換算クラス数という。）が20個であることを示している。換算クラス数は、図７のリソース換算テーブル２２を参照し、20×１＝20として算出することができる。

【００９６】

記憶装置４２は、さらに、個別機能モジュール１３０が受信するイベントを処理するイベントリスナを登録するイベントリスナテーブル４８０を記憶している。

図９は、イベントリスナテーブル４８０のデータ構造を示す図である。

イベントリスナテーブル４８０には、図９に示すように、個別機能モジュール１３０が登録するイベントリスナごとにレコードが登録される。各レコードは、イベントリスナの

名称を登録するフィールド４８２を含んで登録されている。

【００９７】

図３に戻り、ＣＰＵ３０は、マイクロプロセッシングユニット等からなり、ＲＯＭ３２の所定領域に格納されている所定のプログラムを起動させ、そのプログラムに従って、図１０、図１４、図１５および図１７のフローチャートに示す個別機能モジュール制御処理、クラス読込処理、イベントリスナ制御処理およびインスタンス削除処理を共通機能モジュール１２０の処理としてそれぞれ時分割で実行する。

【００９８】

初めに、個別機能モジュール制御処理を図１０を参照しながら詳細に説明する。

図１０は、個別機能モジュール制御処理を示すフローチャートである。

個別機能モジュール制御処理は、個別機能モジュール１３０の削除および実行を制御する処理であって、ＣＰＵ３０において実行されると、図１０に示すように、まず、ステップＳ１００に移行する。

【００９９】

ステップＳ１００では、実行すべき個別機能モジュール１３０の名称および削除すべき個別機能モジュール１３０の名称を実行環境情報登録テーブル４４０から取得し、ステップＳ１０２に移行して、削除すべき個別機能モジュール１３０が存在するか否かを判定し、削除すべき個別機能モジュール１３０が存在すると判定したとき(Yes)は、ステップＳ１０４に移行する。

【０１００】

ステップＳ１０４では、取得した名称に基づいて該当の個別機能モジュール１３０を記憶装置４２から削除し、ステップＳ１０６に移行して、該当の個別機能モジュール１３０に含まれるモジュール情報４２０に基づいて、該当の個別機能モジュール１３０がリソース管理対象であるか否かを判定し、該当の個別機能モジュール１３０がリソース管理対象であると判定したとき(Yes)は、ステップＳ１０８に移行する。

【０１０１】

ステップＳ１０８では、該当の個別機能モジュール１３０に対応するリソース管理テーブル４６０を記憶装置４２から削除し、ステップＳ１１０に移行して、現在起動中のモジュール数を示す変数の値から「１」を減算し、ステップＳ１０２に移行する。

一方、ステップＳ１０６で、該当の個別機能モジュール１３０がリソース管理対象でないと判定したとき(No)は、ステップＳ１０２に移行する。

【０１０２】

一方、ステップＳ１０２で、削除すべき個別機能モジュール１３０が存在しないと判定したとき(No)は、ステップＳ１１２に移行して、実行すべき個別機能モジュール１３０が存在するか否かを判定し、実行すべき個別機能モジュール１３０が存在すると判定したとき(Yes)は、ステップＳ１１４に移行する。

ステップＳ１１４では、現在起動中のモジュール数を示す変数の値が所定の上限値未満であるか否かを判定し、所定の上限値未満であると判定したとき(Yes)は、ステップＳ１１６に移行する。

【０１０３】

ステップＳ１１６では、取得した名称に基づいて該当の個別機能モジュール１３０を記憶装置４２から読み込み、ステップＳ１１８に移行して、読み込んだ個別機能モジュール１３０の実行可否を判定する実行可否判定処理を実行し、ステップＳ１２０に移行する。

ステップＳ１２０では、実行可否判定処理から個別機能モジュール１３０の実行を許可することを示す戻り値が返却されたか否かを判定し、実行を許可することを示す戻り値が返却されたと判定したとき(Yes)は、ステップＳ１２２に移行する。

【０１０４】

ステップＳ１２２では、該当の個別機能モジュール１３０に含まれるモジュール情報４２０に基づいて、該当の個別機能モジュール１３０がリソース管理対象であるか否かを判定し、該当の個別機能モジュール１３０がリソース管理対象であると判定したとき(Yes)

は、ステップＳ１２４に移行する。
ステップＳ１２４では、該当の個別機能モジュール１３０に対応するリソース管理テーブル４６０を生成し、該当の個別機能モジュール１３０に含まれるリソース制限情報４００から上限値を取得し、取得した上限値を、生成したリソース管理テーブル４６０に登録し、ステップＳ１２６に移行して、現在起動中のモジュール数を示す変数の値に「１」を加算し、ステップＳ１２８に移行する。
【０１０５】
ステップＳ１２８では、生成したリソース管理テーブル４６０のアドレスをリソース確保先の参照ポインタに設定し、ステップＳ１３０に移行して、該当の個別機能モジュール１３０を起動するモジュール起動処理を実行し、ステップＳ１３２に移行して、リソース確保先の参照ポインタをクリアし、ステップＳ１１２に移行する。
一方、ステップＳ１２２で、該当の個別機能モジュール１３０がリソース管理対象でないと判定したとき(No)は、ステップＳ１３４に移行して、ステップＳ１３０と同様のモジュール起動処理を実行し、ステップＳ１１２に移行する。
【０１０６】
一方、ステップＳ１２０で、実行可否判定処理から個別機能モジュール１３０の実行を許可しないことを示す戻り値が返却されたと判定したとき(No)は、ステップＳ１１２に移行する。
一方、ステップＳ１１４で、現在起動中のモジュール数を示す変数の値が所定の上限値以上であると判定したとき(No)は、ステップＳ１３６に移行して、モジュール数が上限に達したことを示すログ情報を生成し、一連の処理を終了して元の処理に復帰させる。
【０１０７】
一方、ステップＳ１１２で、実行すべき個別機能モジュール１３０が存在しないと判定したとき(No)は、一連の処理を終了して元の処理に復帰させる。
次に、ステップＳ１１８の実行可否判定処理を図１１を参照しながら詳細に説明する。
図１１は、実行可否判定処理を示すフローチャートである。
実行可否判定処理は、ステップＳ１１８において実行されると、図１１に示すように、まず、ステップＳ２００に移行する。
【０１０８】
ステップＳ２００では、個別機能モジュール１３０に含まれるモジュール情報４２０から機種情報を取得し、ステップＳ２０２に移行して、取得した機種情報と実行環境情報登録テーブル４４０の機種情報が一致しているか否かを判定し、それら機種情報が一致していると判定したとき(Yes)は、ステップＳ２０４に移行する。
ステップＳ２０４では、個別機能モジュール１３０に含まれるモジュール情報４２０から電子署名情報を取得し、ステップＳ２０６に移行して、実行環境情報登録テーブル４４０に基づいて、取得した電子署名情報が対応可能なものであるか否かを判定し、対応可能な電子署名情報であると判定したとき(Yes)は、ステップＳ２０８に移行する。
【０１０９】
ステップＳ２０８では、個別機能モジュール１３０に含まれるモジュール情報４２０に基づいて、個別機能モジュール１３０がリソース管理対象であるか否かを判定し、リソース管理対象であると判定したとき(Yes)は、ステップＳ２０９に移行する。
ステップＳ２０９では、実行環境情報登録テーブル４４０に対応するリソース換算テーブル２２が存在するか否かを判定し、対応のリソース換算テーブル２２が存在すると判定したとき(Yes)は、ステップＳ２１０に移行して、対応のリソース換算テーブル２２を記憶装置４２から読み込み、ステップＳ２１１に移行する。
【０１１０】
ステップＳ２１１では、個別機能モジュール１３０に含まれるリソース制限情報４００から上限値を取得し、ステップＳ２１２に移行して、上限値の取得に成功したか否かを判定し、上限値の取得に成功したと判定したとき(Yes)は、ステップＳ２１４に移行する。
ステップＳ２１４では、取得した上限値が全メモリ残量未満であるか否かを判定し、全

メモリ残量未満であると判定したとき(Yes)は、ステップＳ２１６に移行して、個別機能モジュール１３０の実行を許可することを示す戻り値を返却し、一連の処理を終了して元の処理に復帰させる。

【０１１１】

一方、ステップＳ２１４で、取得した上限値が全メモリ残量以上であると判定したとき(No)は、ステップＳ２１７に移行して、上限値が全メモリ残量以上であることを示すログ情報を生成し、ステップＳ２１８に移行して、個別機能モジュール１３０の実行を許可しないことを示す戻り値を返却し、一連の処理を終了して元の処理に復帰させる。

一方、ステップＳ２１２で、上限値の取得に失敗したと判定したとき(No)、ステップＳ２０６で、対応可能な電子署名情報でないと判定したとき(No)、およびステップＳ２０２で、機種情報が一致しないと判定したとき(No)はいずれも、ステップＳ２１８に移行する。

【０１１２】

一方、ステップＳ２０９で、実行環境情報登録テーブル４４０に対応するリソース換算テーブル２２が存在しないと判定したとき(No)は、ステップＳ２１１に移行する。

一方、ステップＳ２０８で、個別機能モジュール１３０がリソース管理対象でないと判定したとき(No)は、ステップＳ２１６に移行する。

次に、ステップＳ１３０，Ｓ１３４のモジュール起動処理を図１２を参照しながら詳細に説明する。

【０１１３】

図１２は、モジュール起動処理を示すフローチャートである。

モジュール起動処理は、ステップＳ１３０，Ｓ１３４において実行されると、図１２に示すように、まず、ステップＳ３００に移行する。

ステップＳ３００では、個別機能モジュール１３０からクラスを読み込むべきクラス読込命令を出力し、ステップＳ３０２に移行して、クラスの読込に成功したか否かを判定し、クラスの読込に成功したと判定したとき(Yes)は、ステップＳ３０４に移行する。

【０１１４】

ステップＳ３０４では、リソース確保先の参照ポインタが設定されているか否かを判定し、リソース確保先の参照ポインタが設定されていると判定したとき(Yes)は、ステップＳ３０５に移行して、読み込んだクラスの実行に必要なメモリ量を算出し、ステップＳ３０６に移行する。

ステップＳ３０６では、読み込んだクラスが使用するライブラリ等からメモリの使用形態を特定し、特定したメモリの使用形態に対応する換算率を、読み込んだリソース換算テーブル２２から取得し、算出したメモリ量に、取得した換算率を乗算することにより、ネットワークプリンタで使用するメモリ量への換算を行う。

【０１１５】

次いで、ステップＳ３０７に移行して、リソース確保先の参照ポインタが指し示すリソース管理テーブル４６０（以下、参照リソース管理テーブル４６０という。）の使用メモリ量に換算メモリ量を加算し、ステップＳ３０８に移行して、加算した合計のメモリ量が、参照リソース管理テーブル４６０の上限値未満であるか否かを判定し、上限値未満であると判定したとき(Yes)は、ステップＳ３１０に移行する。

【０１１６】

ステップＳ３１０では、読み込んだクラスのインスタンスをメモリ上に生成し、ステップＳ３１２に移行して、リソース確保先の参照ポインタの値を示すリソース確保先参照情報を、生成したインスタンスに保存し、ステップＳ３１３に移行して、換算メモリ量を、生成したインスタンスに保存し、ステップＳ３１４に移行する。

ステップＳ３１４では、インスタンスの生成に成功したか否かを判定し、インスタンスの生成に成功したと判定したとき(Yes)は、ステップＳ３１６に移行して、読み込んだクラスの機能を呼び出すクラス機能呼出処理を実行し、ステップＳ３１８に移行して、個別機能モジュール１３０のイベントリスナを登録するイベントリスナ登録処理を実行し、一

連の処理を終了して元の処理に復帰させる。

【０１１７】

一方、ステップＳ３０８で、合計のメモリ量が上限値以上であると判定したとき(No)は、ステップＳ３２０に移行して、参照リソース管理テーブル４６０の使用メモリ量から換算メモリ量を減算し、ステップＳ３２１に移行する。

ステップＳ３２１では、個別機能モジュール１３０が使用するメモリ量が上限に達したことを示すログ情報を生成し、ステップＳ３２２に移行して、エラーを通知し、ステップＳ３１４に移行する。

【０１１８】

一方、ステップＳ３０４で、リソース確保先の参照ポインタが設定されていないと判定したとき(No)は、ステップＳ３２４に移行して、読み込んだクラスのインスタンスをメモリ上に生成し、ステップＳ３１４に移行する。

一方、ステップＳ３１４で、インスタンスの生成に失敗したと判定したとき(No)、およびステップＳ３０２で、クラスの読込に失敗したと判定したとき(No)はいずれも、ステップＳ３１８に移行する。

【０１１９】

次に、ステップＳ３１８のイベントリスナ登録処理を図１３を参照しながら詳細に説明する。

図１３は、イベントリスナ登録処理を示すフローチャートである。

イベントリスナ登録処理は、ステップＳ３１８において実行されると、図１３に示すように、まず、ステップＳ４００に移行する。

【０１２０】

ステップＳ４００では、個別機能モジュール１３０からイベントリスナクラスを読み込むべきクラス読込命令を出力し、ステップＳ４０２に移行して、イベントリスナクラスの読込に成功したか否かを判定し、イベントリスナクラスの読込に成功したと判定したとき(Yes)は、ステップＳ４０４に移行する。

ステップＳ４０４では、リソース確保先の参照ポインタが設定されているか否かを判定し、リソース確保先の参照ポインタが設定されていると判定したとき(Yes)は、ステップＳ４０５に移行して、読み込んだイベントリスナクラスの実行に必要なメモリ量を算出し、ステップＳ４０６に移行する。

【０１２１】

ステップＳ４０６では、読み込んだイベントリスナクラスが使用するライブラリ等からメモリの使用形態を特定し、特定したメモリの使用形態に対応する換算率を、読み込んだリソース換算テーブル２２から取得し、算出したメモリ量に、取得した換算率を乗算することにより、ネットワークプリンタで使用するメモリ量への換算を行う。

次いで、ステップＳ４０７に移行して、参照リソース管理テーブル４６０の使用メモリ量に換算メモリ量を加算し、ステップＳ４０８に移行して、加算した合計のメモリ量が、参照リソース管理テーブル４６０の上限値未満であるか否かを判定し、上限値未満であると判定したとき(Yes)は、ステップＳ４１０に移行する。

【０１２２】

ステップＳ４１０では、読み込んだイベントリスナクラスのインスタンスをメモリ上に生成し、ステップＳ４１２に移行して、リソース確保先の参照ポインタの値を示すリソース確保先参照情報を、生成したインスタンスに保存し、ステップＳ４１３に移行して、換算メモリ量を、生成したインスタンスに保存し、ステップＳ４１４に移行する。

ステップＳ４１４では、インスタンスの生成に成功したか否かを判定し、インスタンスの生成に成功したと判定したとき(Yes)は、ステップＳ４１６に移行して、生成したインスタンスのイベントリスナをイベントリスナ実行リストに登録し、一連の処理を終了して元の処理に復帰させる。

【０１２３】

一方、ステップＳ４０８で、合計のメモリ量が上限値以上であると判定したとき(No)は

、ステップＳ４１８に移行して、参照リソース管理テーブル４６０の使用メモリ量から換算メモリ量を減算し、ステップＳ４１９に移行する。
ステップＳ４１９では、個別機能モジュール１３０が使用するメモリ量が上限に達したことを示すログ情報を生成し、ステップＳ４２０に移行して、エラーを通知し、ステップＳ４１４に移行する。
【０１２４】
一方、ステップＳ４０４で、リソース確保先の参照ポインタが設定されていないと判定したとき(No)は、ステップＳ４２２に移行して、読み込んだイベントリスナクラスのインスタンスをメモリ上に生成し、ステップＳ４１４に移行する。
一方、ステップＳ４１４で、インスタンスの生成に失敗したと判定したとき(No)、およびステップＳ４０２で、イベントリスナクラスの読込に失敗したと判定したとき(No)はいずれも、一連の処理を終了して元の処理に復帰させる。
【０１２５】
次に、クラス読込処理を図１４を参照しながら詳細に説明する。
図１４は、クラス読込処理を示すフローチャートである。
クラス読込処理は、クラス読込命令に応じてクラスを読み込む処理であって、ＣＰＵ３０において実行されると、図１４に示すように、まず、ステップＳ５００に移行する。
ステップＳ５００では、クラス読込命令を取得したか否かを判定し、クラス読込命令を取得したと判定したとき(Yes)は、ステップＳ５０２に移行するが、そうでないと判定したとき(No)は、クラス読込命令を取得するまでステップＳ５００で待機する。
【０１２６】
ステップＳ５０２では、クラス読込命令に係るクラスがキャッシュテーブルに登録されているか否かを判定し、キャッシュテーブルに登録されていないと判定したとき(No)は、ステップＳ５０４に移行する。
ステップＳ５０４では、クラス読込命令に係るクラスが属する個別機能モジュール１３０を特定し、ステップＳ５０６に移行して、特定した該当の個別機能モジュール１３０に含まれるモジュール情報４２０に基づいて、該当の個別機能モジュール１３０がリソース管理対象であるか否かを判定し、リソース管理対象であると判定したとき(Yes)は、ステップＳ５０８に移行する。
【０１２７】
ステップＳ５０８では、該当の個別機能モジュール１３０に対応するリソース管理テーブル４６０のアドレスをリソース確保先の参照ポインタに設定し、ステップＳ５０９に移行する。
ステップＳ５０９では、クラス数に対応する換算率を、読み込んだリソース換算テーブル２２から取得し、クラス読込命令に係るクラスの数「１」に、取得した換算率を乗算することにより、ネットワークプリンタで起動するクラス数への換算を行う。
【０１２８】
次いで、ステップＳ５１０に移行して、参照リソース管理テーブル４６０の起動クラス数に換算クラス数を加算し、ステップＳ５１２に移行して、加算した合計のクラス数が、参照リソース管理テーブル４６０の上限値未満であるか否かを判定し、上限値未満であると判定したとき(Yes)は、ステップＳ５１４に移行する。
ステップＳ５１４では、クラス読込命令に係るクラスを個別機能モジュール１３０から読み込み、ステップＳ５１６に移行して、読み込んだクラスをキャッシュテーブルに登録し、ステップＳ５１７に移行する。
【０１２９】
ステップＳ５１７では、換算クラス数を、読み込んだクラスに保存し、ステップＳ５１８に移行して、リソース確保先の参照ポインタをクリアし、一連の処理を終了して元の処理に復帰させる。
一方、ステップＳ５１２で、合計のクラス数が上限値以上であると判定したとき(No)は、ステップＳ５２０に移行して、参照リソース管理テーブル４６０の起動クラス数から換

算クラス数を減算し、ステップＳ５２１に移行する。
【０１３０】
ステップＳ５２１では、個別機能モジュール１３０が起動するクラス数が上限に達したことを示すログ情報を生成し、ステップＳ５２２に移行して、エラーを通知し、ステップＳ５１８に移行する。
一方、ステップＳ５０６で、該当の個別機能モジュール１３０がリソース管理対象でないと判定したとき(No)は、ステップＳ５２４に移行して、クラス読込命令に係るクラスを個別機能モジュール１３０から読み込み、ステップＳ５２６に移行して、読み込んだクラスをキャッシュテーブルに登録し、一連の処理を終了して元の処理に復帰させる。
【０１３１】
一方、ステップＳ５０２で、クラス読込命令に係るクラスがキャッシュテーブルに登録されていると判定したとき(Yes)は、一連の処理を終了して元の処理に復帰させる。
次に、イベントリスナ制御処理を図１５を参照しながら詳細に説明する。
図１５は、イベントリスナ制御処理を示すフローチャートである。
イベントリスナ制御処理は、イベントリスナの実行を制御する処理であって、ＣＰＵ３０において実行されると、図１５に示すように、まず、ステップＳ６００に移行する。
【０１３２】
ステップＳ６００では、イベントリスナ実行リストを取得し、ステップＳ６０２に移行して、取得したイベントリスナ実行リストに基づいて、実行すべきイベントリスナが存在するか否かを判定し、実行すべきイベントリスナが存在すると判定したとき(Yes)は、ステップＳ６０４に移行する。
ステップＳ６０４では、該当のイベントリスナの生成元となった個別機能モジュール１３０を特定し、ステップＳ６０６に移行して、特定した該当の個別機能モジュール１３０に含まれるモジュール情報４２０に基づいて、該当の個別機能モジュール１３０がリソース管理対象であるか否かを判定し、リソース管理対象であると判定したとき(Yes)は、ステップＳ６０８に移行する。
【０１３３】
ステップＳ６０８では、該当の個別機能モジュール１３０に対応するリソース管理テーブル４６０のアドレスをリソース確保先の参照ポインタに設定し、ステップＳ６１０に移行して、該当のイベントリスナを実行するイベントリスナ実行処理を実行し、ステップＳ６１２に移行して、リソース確保先の参照ポインタをクリアし、ステップＳ６１４に移行する。
【０１３４】
ステップＳ６１４では、該当のイベントリスナをイベントリスナ実行リストから削除し、ステップＳ６０２に移行する。
一方、ステップＳ６０６で、該当の個別機能モジュール１３０がリソース管理対象でないと判定したとき(No)は、ステップＳ６１６に移行して、該当のイベントリスナを実行し、ステップＳ６１４に移行する。
【０１３５】
一方、ステップＳ６０２で、実行すべきイベントリスナが存在しないと判定したとき(No)は、一連の処理を終了して元の処理に復帰させる。
次に、ステップＳ６１０のイベントリスナ実行処理を図１６を参照しながら詳細に説明する。
図１６は、イベントリスナ実行処理を示すフローチャートである。
【０１３６】
イベントリスナ実行処理は、ステップＳ６１０において実行されると、図１６に示すように、まず、ステップＳ７００に移行する。
ステップＳ７００では、イベントリスナに含まれる命令リストの先頭にプログラムポインタを移動し、ステップＳ７０２に移行して、プログラムポインタが指し示すアドレスに実行すべき命令が存在するか否かを判定し、実行すべき命令が存在すると判定したとき(Y

es)は、ステップＳ７０３に移行して、命令の実行に必要なメモリ量を算出し、ステップＳ７４０に移行する。

【０１３７】

ステップＳ７０４では、命令の実行に使用するライブラリ等からメモリの使用形態を特定し、特定したメモリの使用形態に対応する換算率を、読み込んだリソース換算テーブル２２から取得し、算出したメモリ量に、取得した換算率を乗算することにより、ネットワークプリンタで使用するメモリ量への換算を行う。

次いで、ステップＳ７０５に移行して、参照リソース管理テーブル４６０の使用メモリ量に換算メモリ量を加算し、ステップＳ７０６に移行して、加算した合計のメモリ量が、参照リソース管理テーブル４６０の上限値未満であるか否かを判定し、上限値未満であると判定したとき(Yes)は、ステップＳ７０８に移行する。

【０１３８】

ステップＳ７０８では、メモリを確保し、ステップＳ７１０に移行して、プログラムポインタが指し示すアドレスの命令を実行し、ステップＳ７１２に移行して、イベントリスナに含まれる命令リストの次にプログラムポインタを移動し、ステップＳ７１３に移行して、所定の待機時間だけ処理を待機し、ステップＳ７０２に移行する。この待機時間の決定については、後段の実施の形態で詳述する。

【０１３９】

一方、ステップＳ７０６で、合計のメモリ量が上限値以上であると判定したとき(No)は、ステップＳ７１４に移行して、参照リソース管理テーブル４６０の使用メモリ量から換算メモリ量を減算し、ステップＳ７１５に移行する。

ステップＳ７１５では、個別機能モジュール１３０が使用するメモリ量が上限に達したことを示すログ情報を生成し、ステップＳ７１６に移行して、エラーを通知し、ステップＳ７１２に移行する。

【０１４０】

一方、ステップＳ７０２で、実行すべき命令が存在しないと判定したとき(No)は、一連の処理を終了して元の処理に復帰させる。

次に、インスタンス削除処理を図１７を参照しながら詳細に説明する。

図１７は、インスタンス削除処理を示すフローチャートである。

インスタンス削除処理は、インスタンスを削除する処理であって、ＣＰＵ３０において実行されると、図１７に示すように、まず、ステップＳ８００に移行する。

【０１４１】

ステップＳ８００では、削除すべきインスタンスを登録したインスタンス削除リストを取得し、ステップＳ８０２に移行して、取得したインスタンス削除リストに基づいて、削除すべきインスタンスが存在するか否かを判定し、削除すべきインスタンスが存在すると判定したとき(Yes)は、ステップＳ８０４に移行する。

ステップＳ８０４では、該当のインスタンスからリソース確保先参照情報を取得し、ステップＳ８０５に移行して、リソース確保先参照情報の取得に成功したか否かを判定し、リソース確保先参照情報の取得に成功したと判定したとき(Yes)は、ステップＳ８０６に移行する。

【０１４２】

ステップＳ８０６では、取得したリソース確保先参照情報に基づいてリソース確保先の参照ポインタを設定し、ステップＳ８０８に移行して、該当のインスタンスを削除し、ステップＳ８１０に移行して、該当のインスタンスの実行に必要なメモリ量を参照リソース管理テーブル４６０の使用メモリ量から減算し、ステップＳ８１２に移行する。

ステップＳ８１２では、リソース確保先の参照ポインタをクリアし、ステップＳ８１４に移行して、該当のインスタンスをインスタンス削除リストから削除し、ステップＳ８０２に移行する。

【０１４３】

一方、ステップＳ８０５で、リソース確保先参照情報の取得に失敗したと判定したとき

(No)は、ステップＳ８１６に移行して、該当のインスタンスを削除し、ステップＳ８１４に移行する。

一方、ステップＳ８０２で、削除すべきインスタンスが存在しないと判定したとき(No)は、一連の処理を終了して元の処理に復帰させる。

【０１４４】

次に、本実施の形態の動作を説明する。

初めに、リソース管理対象となる個別機能モジュール１３０を実行する場合を説明する。

ホスト端末１００では、共通機能モジュール１２０の実行により個別機能モジュール制御処理が実行される。個別機能モジュール制御処理では、ステップＳ１０２～Ｓ１１０を経て、削除すべき個別機能モジュール１３０が存在する場合は、その個別機能モジュール１３０が削除される。次いで、ステップＳ１１４を経て、現在起動中のモジュール数が所定の上限値未満であると判定されると、ステップＳ１１６，Ｓ１１８を経て、該当の個別機能モジュール１３０が読み込まれ、読み込まれた個別機能モジュール１３０の実行可否が判定される。実行可否判定処理では、個別機能モジュール１３０について、一致する機種情報および対応可能な電子署名情報を有し、かつ、使用可能なメモリ量の上限値が全メモリ残量未満である場合に実行が許可される。

【０１４５】

個別機能モジュール１３０の実行が許可されると、ステップＳ１２４～Ｓ１２８を経て、リソース管理テーブル４６０が生成され、現在起動中のモジュール数が「１」加算され、個別機能モジュール１３０が起動する。モジュール起動処理では、ステップＳ５０９，Ｓ５１０，Ｓ３０５～Ｓ３０７を経て、個別機能モジュール１３０の起動クラス数および使用メモリ量が、ネットワークプリンタでの起動クラス数および使用メモリ量に換算されて加算される。このとき、起動クラス数および使用メモリ量のいずれかが上限値以上となると、ステップＳ５２１，Ｓ５２２またはステップＳ３２１，Ｓ３２２を経て、個別機能モジュール１３０の起動クラス数または使用メモリ量が上限に達したことを示すログ情報が生成されるとともにエラーが通知され、クラスの読込またはインスタンスの生成が中止される。

【０１４６】

これに対し、起動クラス数および使用メモリ量のいずれも上限値未満である場合は、ステップＳ５１４，Ｓ３１０，Ｓ３１８を経て、個別機能モジュール１３０のクラスが読み込まれ、読み込まれたクラスのインスタンスが生成され、個別機能モジュール１３０のイベントリスナが登録される。イベントリスナ登録処理では、ステップＳ５０９，Ｓ５１０，Ｓ４０５～Ｓ４０７を経て、個別機能モジュール１３０の起動クラス数および使用メモリ量が、ネットワークプリンタでの起動クラス数および使用メモリ量に換算されて加算される。このとき、起動クラス数および使用メモリ量のいずれかが上限値以上となると、ステップＳ５２１，Ｓ５２２またはステップＳ４１９，Ｓ４２０を経て、個別機能モジュール１３０の起動クラス数または使用メモリ量が上限に達したことを示すログ情報が生成されるとともにエラーが通知され、イベントリスナクラスの読込またはインスタンスの生成が中止される。

【０１４７】

これに対し、起動クラス数および使用メモリ量のいずれも上限値未満である場合は、ステップＳ５１４，Ｓ４１０，Ｓ４１６を経て、イベントリスナクラスが読み込まれ、イベントリスナクラスのインスタンスが生成され、生成されたインスタンスのイベントリスナがイベントリスナ実行リストに登録される。

一方、ホスト端末１００では、共通機能モジュール１２０の実行によりイベントリスナ制御処理が実行される。イベントリスナ制御処理では、ステップＳ７０３～Ｓ７０５を経て、実行すべきイベントリスナの生成元となった個別機能モジュール１３０の使用メモリ量が、ネットワークプリンタでの使用メモリ量に換算されて加算される。このとき、使用メモリ量が上限値以上となると、ステップＳ７１５，Ｓ７１６を経て、個別機能モジュー

ル１３０が使用するメモリ量が上限に達したことを示すログ情報が生成されるとともにエラーが通知され、イベントリスナの実行が中止される。

【０１４８】

これに対し、使用メモリ量が上限値未満である場合は、ステップＳ７１０を経て、イベントリスナに含まれる命令が実行される。

一方、ホスト端末１００では、共通機能モジュール１２０の実行によりインスタンス削除処理が実行される。インスタンス削除処理では、削除すべきインスタンスが存在する場合は、ステップＳ８０８，Ｓ８１０を経て、そのインスタンスが削除され、そのイベントリスナの生成元となった個別機能モジュール１３０の使用メモリ量が減算される。

【０１４９】

図１８は、エラーが発生した場合のログファイルの内容を示す図である。

エラーが発生した場合、ログファイルには、図１８に示すように、個別機能モジュール１３０が起動および停止したことを示すログ情報のほか、個別機能モジュール１３０の起動クラス数または使用メモリ量が上限に達したことを示すログ情報が記録される。

図１９は、エラーが発生しない場合のログファイルの内容を示す図である。

【０１５０】

これに対し、エラーが発生しない場合、ログファイルには、図１９に示すように、個別機能モジュール１３０が起動および停止したことを示すログ情報のみが記録される。

次に、リソース管理対象でない個別機能モジュール１３０を実行する場合を説明する。

ホスト端末１００では、個別機能モジュール制御処理が実行されると、ステップＳ１１６，Ｓ１１８を経て、該当の個別機能モジュール１３０が読み込まれ、読み込まれた個別機能モジュール１３０の実行可否が判定される。

【０１５１】

個別機能モジュール１３０の実行が許可されると、ステップＳ１３４を経て、個別機能モジュール１３０が起動する。モジュール起動処理では、ステップＳ５２４，Ｓ３２４，Ｓ３１８を経て、個別機能モジュール１３０のクラスが読み込まれ、読み込まれたクラスのインスタンスが生成され、個別機能モジュール１３０のイベントリスナが登録される。イベントリスナ登録処理では、ステップＳ５２４，Ｓ４２２，Ｓ４１６を経て、イベントリスナクラスが読み込まれ、イベントリスナクラスのインスタンスが生成され、生成されたインスタンスのイベントリスナがイベントリスナ実行リストに登録される。

【０１５２】

一方、ホスト端末１００では、イベントリスナ制御処理が実行されると、ステップＳ６１６を経て、実行すべきイベントリスナに含まれる命令が実行される。

一方、ホスト端末１００では、インスタンス削除処理が実行されると、削除すべきインスタンスが存在する場合は、ステップＳ８１６を経て、そのインスタンスが削除される。

図２０は、リソース管理対象となる個別機能モジュールｂ，ｃを並列に実行した場合を示すタイムチャートである。

【０１５３】

図２０において、実線は、個別機能モジュールｂのスレッド、および共通機能モジュール１２０のスレッドのうち個別機能モジュールｂの実行に用いられるものを示している。また、一点鎖線は、個別機能モジュールｃのスレッド、および共通機能モジュール１２０のスレッドのうち個別機能モジュールｃの実行に用いられるものを示している。

個別機能モジュールｂが実行されると、共通機能モジュール１２０のＡＭスレッド（起動処理部）が実行され、個別機能モジュールｂが起動してそのスレッドが実行される。また、共通機能モジュール１２０のＡＭスレッドが実行され、個別機能モジュールｂのイベントリスナが生成される。そして、個別機能モジュールｂに対応するイベントが発生すると、共通機能モジュール１２０のＡＭスレッド（イベント処理部）が実行され、個別機能モジュールｂのクラスが読み込まれ、読み込まれたインスタンスが生成される。個別機能モジュールｂが不要となった場合は、共通機能モジュール１２０のインスタンス削除スレッドが実行され、個別機能モジュールｂのインスタンスが削除される。この一連の処理に

おいて、共通機能モジュール１２０および個別機能モジュールｂのスレッドの実行に伴って増減する起動クラス数および使用メモリ量は、個別機能モジュールｂが使用するリソースの量として管理され、個別機能モジュールｂに設定された所定の上限値未満となるように制限される。

【０１５４】

この動作は、個別機能モジュールｃについても同様である。ただし、その一連の処理において、共通機能モジュール１２０および個別機能モジュールｃのスレッドの実行に伴って増減する起動クラス数および使用メモリ量は、個別機能モジュールｃが使用するリソースの量として管理され、個別機能モジュールｃに設定された所定の上限値未満となるように制限される。

【０１５５】

このようにして、本実施の形態では、個別機能モジュール１３０がホスト端末１００で使用するリソースの量を測定し、測定したリソースの量を、ネットワークプリンタで使用するリソースの量に換算し、個別機能モジュール１３０から上限値を取得し、換算したリソースの量および取得した上限値に基づいて、個別機能モジュール１３０が使用するリソースの量が上限に達したことを示すログ情報を生成するようになっている。

【０１５６】

これにより、個別機能モジュール１３０がネットワークプリンタで使用するリソースの量が、リソースの上限値に達するかをネットワークプリンタへの導入前に検証することができる。したがって、従来に比して、ソフトウェアの開発を容易に行うことができるとともに安定性の高いソフトウェアを開発することができる。

さらに、本実施の形態では、リソース管理対象となる各個別機能モジュール１３０ごとに、その個別機能モジュール１３０が使用するメモリ量、並びに共通機能モジュール１２０がその個別機能モジュール１３０の実行に使用するメモリ量および起動するクラス数を測定するようになっている。

【０１５７】

これにより、共通機能モジュール１２０がネットワークプリンタで使用するリソースの量が、リソースの上限値に達するかを個別機能モジュール１３０単位で検証することができる。

さらに、本実施の形態では、換算したリソースの量および取得した上限値に基づいて個別機能モジュール１３０によるリソースの確保を禁止するようになっている。

【０１５８】

これにより、個別機能モジュール１３０が上限値を超えてリソースの量を使用するのを制限することができる。

さらに、本実施の形態では、ホスト端末１００で所定条件で使用されるリソースの量およびネットワークプリンタで同所定条件で使用されるリソースの量に基づき決定される換算率を登録したリソース換算テーブル２２に基づいて換算を行うようになっている。

【０１５９】

これにより、ホスト端末１００およびネットワークプリンタの間でリソースの量を比較的正確に換算することができる。

さらに、本実施の形態では、共通機能モジュール１２０および個別機能モジュール１３０が使用するリソースの種別に応じてリソース換算テーブル２２から対応する換算率を取得し、取得した換算率に基づいて換算を行うようになっている。

【０１６０】

これにより、共通機能モジュール１２０および個別機能モジュール１３０が使用するリソースの種別に応じた換算を行うことができるので、ホスト端末１００およびネットワークプリンタの間でリソースの量をさらに正確に換算することができる。

さらに、本実施の形態では、共通機能モジュール１２０および個別機能モジュール１３０が使用するリソースの形態に応じてリソース換算テーブル２２から対応する換算率を取得し、取得した換算率に基づいて換算を行うようになっている。

【０１６１】
これにより、共通機能モジュール１２０および個別機能モジュール１３０が使用するリソースの形態に応じた換算を行うことができるので、ホスト端末１００およびネットワークプリンタの間でリソースの量をさらに正確に換算することができる。
さらに、本実施の形態では、リソース換算テーブル２２は、複数のテストモジュールのそれぞれについて、ホスト端末１００およびネットワークプリンタでそのテストモジュールが使用するリソースの量に基づいて換算率を決定し、各テストモジュールについて決定された換算率のうち最大値を登録した。
【０１６２】
これにより、リソース換算テーブル２２には、各テストモジュールについて決定された換算率のうち最大値が登録されているので、共通機能モジュール１２０および個別機能モジュール１３０が使用するリソースの量を多めに見積もることができる。したがって、個別機能モジュール１３０の動作を比較的確実に保証することができる。
次に、本発明の第２の実施の形態を図面を参照しながら説明する。図２１ないし図２４は、本発明に係るリソース管理システムおよび、並びにの第２の実施の形態を示す図である。
【０１６３】
本実施の形態は、本発明に係るリソース管理システムおよび、並びにを、図２１に示すように、ホスト端末１００上のＪＡＶＡ（登録商標）アプリケーションの実行環境において、ネットワークプリンタの動作を制御するためのＪＡＶＡ（登録商標）クラスセットをエミュレーションする場合について適用したものであり、上記第１の実施の形態と異なるのは、リソースの上限値を換算する点にある。なお、以下、上記第１の実施の形態と異なる部分についてのみ説明し、上記第１の実施の形態と重複する部分については同一の符号を付して説明を省略する。
【０１６４】
まず、本発明を適用するホスト端末１００の機能概要を説明する。
図２１は、ホスト端末１００の機能概要を示す機能ブロック図である。
共通機能モジュール１２０は、図２１に示すように、個別機能モジュール管理部１４、リソース測定部１６、上限値取得部１８、リソース制限部２０、リソース換算テーブル２２およびログ情報生成部２６のほか、リソースの量を換算するリソース換算部２８を有して構成されている。
【０１６５】
リソース換算部２８は、リソース換算テーブル２２に基づいて、上限値取得部１８で取得したリソースの量を、ホスト端末１００でのリソースの上限値に換算する。
リソース制限部２０は、リソース測定部１６で測定したリソースの量が、リソース換算部２８で換算した上限値未満となるように、個別機能モジュール１３０が使用するリソースの量、および個別機能モジュール管理部１４がその個別機能モジュール１３０の実行に使用するリソースの量を制限する。
【０１６６】
ログ情報生成部２６は、リソース測定部１６で測定したリソースの量が、リソース換算部２８で換算した上限値以上であると判定したときは、個別機能モジュール１３０が使用するリソースの量が上限に達したことを示すログ情報を生成する。
次に、ホスト端末１００の構成を詳細に説明する。
記憶装置４２は、図７のリソース換算テーブル２２に代えて、図２２のリソース換算テーブル２２を記憶している。
【０１６７】
図２２は、リソース換算テーブル２２のデータ構造を示す図である。
リソース換算テーブル２２には、図２２に示すように、リソースの種別または使用形態ごとに１つのレコードが登録されている。各レコードは、リソースの名称を登録するフィールド５０２と、換算率を登録するフィールド５０４とを含んで構成されている。

図２２の例では、第１段目のレコードには、リソースの名称として「メモリ」が、換算率として「１」がそれぞれ登録されている。これは、個別機能モジュール１３０がネットワークプリンタでのメモリ量の上限値を換算率「１」で除算することにより、ホスト端末１００でのメモリ量の上限値への換算を行うことを示している。

【０１６８】

また、第２段目のレコードには、リソースの名称として「クラス数」が、換算率として「１」がそれぞれ登録されている。これは、個別機能モジュール１３０がネットワークプリンタで起動可能なクラス数の上限値を換算率「１」で除算することにより、ホスト端末１００で起動可能なクラス数への換算を行うことを示している。

なお、リソース換算テーブル２２には、複数のテストモジュールのそれぞれについて、ホスト端末１００およびネットワークプリンタでそのテストモジュールが使用するリソースの量に基づいて換算率を決定し、各テストモジュールについて決定された換算率のうち最大値が登録されている。リソース換算テーブル２２の生成については、後段の実施の形態で詳述する。

【０１６９】

記憶装置４２は、さらに、図８のリソース管理テーブル４６０に代えて、図２３のリソース管理テーブル４６０を記憶している。

図２３は、リソース管理テーブル４６０のデータ構造を示す図である。

リソース管理テーブル４６０には、図２３に示すように、リソースの種別ごとに１つのレコードが登録されている。各レコードは、リソースの名称を登録するフィールド４６２と、個別機能モジュール１３０がネットワークプリンタでのリソースの上限値を登録するフィールド４６４と、フィールド４６４の値を、ホスト端末１００でのリソースの上限値に換算した値を登録するフィールド４７０と、個別機能モジュール１３０がホスト端末１００で現在使用しているリソースの量を登録するフィールド４６６とを含んで登録されている。

【０１７０】

図２３の例では、第１段目のレコードには、上限値として「1000000」が、換算値として「666666」がそれぞれ登録されている。これは、個別機能モジュール１３０がネットワークプリンタでのメモリ量の上限値が1000000[byte]であり、ホスト端末１００でのメモリ量の上限値に換算した値（以下、換算メモリ上限値という。）が666666[byte]であることを示している。換算メモリ上限値は、図２２のリソース換算テーブル２２を参照し、1000000／1.5＝666666として算出することができる。

【０１７１】

また、第２段目のレコードには、上限値として「100」が、換算値として「20」がそれぞれ登録されている。これは、個別機能モジュール１３０がネットワークプリンタで起動可能なクラス数の上限値が100個であり、ホスト端末１００で起動可能なクラス数の上限値に換算した値（以下、換算クラス上限値という。）が20個であることを示している。換算クラス上限値は、図２２のリソース換算テーブル２２を参照し、20／１＝20として算出することができる。

【０１７２】

次に、ホスト端末１００で実行される処理を説明する。

ＣＰＵ３０は、図１１の実行可否判定処理に代えて、図２４のフローチャートに示す実行可否判定処理を実行する。

図２４は、実行可否判定処理を示すフローチャートである。

実行可否判定処理は、ステップＳ１１８において実行されると、図２４に示すように、まず、ステップＳ２００～Ｓ２１２を経てステップＳ２１３に移行する。

【０１７３】

ステップＳ２１３では、取得した上限値に対応する換算率を、読み込んだリソース換算テーブル２２から取得し、取得した上限値を、取得した換算率で除算することにより、ホスト端末１００での上限値への換算を行う。

次いで、ステップＳ２１４に移行して、換算メモリ上限値が全メモリ残量未満であるか否かを判定し、全メモリ残量未満であると判定したとき(Yes)は、ステップＳ２１６に移行するが、そうでないと判定したとき(No)は、ステップＳ２１７に移行する。

【０１７４】

なお、上記第１の実施の形態においては、ステップＳ３０６，Ｓ４０６，Ｓ５０９，Ｓ７０４の処理を設けたが、本実施の形態では、それらの処理が不要となる。

また、上記第１の実施の形態においては、ステップＳ３１３，Ｓ３２０，Ｓ４１８，Ｓ５１７，Ｓ５２０，Ｓ７１４の処理で換算メモリ量および換算クラス数を取り扱ったが、本実施の形態では、ホスト端末１００での使用メモリ量および起動クラス数を取り扱う。

【０１７５】

また、上記第１の実施の形態においては、ステップＳ３０８，Ｓ４０８，Ｓ５１２，Ｓ７０６でネットワークプリンタでの使用メモリ量および起動クラス数を取り扱ったが、本実施の形態では、換算メモリ上限値および換算メモリを取り扱う。

次に、本実施の形態の動作を説明する。

初めに、リソース管理対象となる個別機能モジュール１３０を実行する場合を説明する。

【０１７６】

ホスト端末１００では、共通機能モジュール１２０の実行により個別機能モジュール制御処理が実行される。個別機能モジュール制御処理では、ステップＳ１０２～Ｓ１１０を経て、削除すべき個別機能モジュール１３０が存在する場合は、その個別機能モジュール１３０が削除される。次いで、ステップＳ１１４を経て、現在起動中のモジュール数が所定の上限値未満であると判定されると、ステップＳ１１６，Ｓ１１８を経て、該当の個別機能モジュール１３０が読み込まれ、読み込まれた個別機能モジュール１３０の実行可否が判定される。実行可否判定処理では、ステップＳ２１３を経て、リソース制限情報４００から取得された上限値が、ホスト端末１００での上限値に換算される。そして、個別機能モジュール１３０について、一致する機種情報および対応可能な電子署名情報を有し、かつ、換算メモリ上限値が全メモリ残量未満である場合に実行が許可される。

【０１７７】

個別機能モジュール１３０の実行が許可されると、ステップＳ１２４～Ｓ１２８を経て、リソース管理テーブル４６０が生成され、現在起動中のモジュール数が「１」加算され、個別機能モジュール１３０が起動する。モジュール起動処理では、ステップＳ５１０，Ｓ３０５，Ｓ３０７を経て、個別機能モジュール１３０の起動クラス数および使用メモリ量が加算される。このとき、起動クラス数および使用メモリ量のいずれかが換算上限値（換算クラス数または換算メモリ上限値をいう。以下、同様。）以上となると、ステップＳ５２１，Ｓ５２２またはステップＳ３２１，Ｓ３２２を経て、個別機能モジュール１３０の起動クラス数または使用メモリ量が上限に達したことを示すログ情報が生成されるとともにエラーが通知され、クラスの読込またはインスタンスの生成が中止される。

【０１７８】

これに対し、起動クラス数および使用メモリ量のいずれも換算上限値未満である場合は、ステップＳ５１４，Ｓ３１０，Ｓ３１８を経て、個別機能モジュール１３０のクラスが読み込まれ、読み込まれたクラスのインスタンスが生成され、個別機能モジュール１３０のイベントリスナが登録される。イベントリスナ登録処理では、ステップＳ５１０，Ｓ４０５，Ｓ４０７を経て、個別機能モジュール１３０の起動クラス数および使用メモリ量が加算される。このとき、起動クラス数および使用メモリ量のいずれかが換算上限値以上となると、ステップＳ５２１，Ｓ５２２またはステップＳ４１９，Ｓ４２０を経て、個別機能モジュール１３０の起動クラス数または使用メモリ量が上限に達したことを示すログ情報が生成されるとともにエラーが通知され、イベントリスナクラスの読込またはインスタンスの生成が中止される。

【０１７９】

これに対し、起動クラス数および使用メモリ量のいずれも換算上限値未満である場合は

、ステップＳ５１４，Ｓ４１０，Ｓ４１６を経て、イベントリスナクラスが読み込まれ、イベントリスナクラスのインスタンスが生成され、生成されたインスタンスのイベントリスナがイベントリスナ実行リストに登録される。

一方、ホスト端末１００では、共通機能モジュール１２０の実行によりイベントリスナ制御処理が実行される。イベントリスナ制御処理では、ステップＳ７０３，Ｓ７０５を経て、実行すべきイベントリスナの生成元となった個別機能モジュール１３０の使用メモリ量が加算される。このとき、使用メモリ量が換算上限値以上となると、ステップＳ７１５，Ｓ７１６を経て、個別機能モジュール１３０が使用するメモリ量が上限に達したことを示すログ情報が生成されるとともにエラーが通知され、イベントリスナの実行が中止される。

【０１８０】

これに対し、使用メモリ量が換算上限値未満である場合は、ステップＳ７１０を経て、イベントリスナに含まれる命令が実行される。

このようにして、本実施の形態では、個別機能モジュール１３０がホスト端末１００で使用するリソースの量を測定し、個別機能モジュール１３０から上限値を取得し、取得した上限値を、ホスト端末１００でのリソースの上限値に換算し、測定したリソースの量および換算した上限値に基づいて、個別機能モジュール１３０が使用するリソースの量が上限に達したことを示すログ情報を生成するようになっている。

【０１８１】

これにより、個別機能モジュール１３０がネットワークプリンタで使用するリソースの量が、リソースの上限値に達するかをネットワークプリンタへの導入前に検証することができる。したがって、従来に比して、ソフトウェアの開発を容易に行うことができるとともに安定性の高いソフトウェアを開発することができる。

上記第２の実施の形態において、リソース測定部１６およびステップＳ３０５，Ｓ４０５，Ｓ７０３は、形態１ないし５のリソース測定手段に対応し、ステップＳ３０５，Ｓ４０５，Ｓ７０３は、形態１３ないし１７、２５ないし３２のリソース測定ステップに対応し、リソース換算部２８およびステップＳ２１３は、形態１、２、４ないし８のリソース換算手段に対応している。また、ステップＳ２１３は、形態１３、１４、１６ないし２０、２５ないし２８、３０ないし３５のリソース換算ステップに対応し、上限値取得部１８およびステップＳ２１１は、形態１、２または４のリソース上限値取得手段に対応し、ステップＳ２１１は、形態１３、１４、１６、２５ないし２８、３０または３１のリソース上限値取得ステップに対応している。

【０１８２】

また、上記第２の実施の形態において、ログ情報生成部２６およびステップＳ３２１，Ｓ４１９，Ｓ５２１，Ｓ７１５は、形態１、２、４または１２のリソース制限通知手段に対応し、ステップＳ３２１，Ｓ４１９，Ｓ５２１，Ｓ７１５は、形態１３、１４、１６、２４ないし２８、３０、３１または３９のリソース制限通知ステップに対応している。また、リソース制限部２０およびステップＳ３０８，Ｓ４０８，Ｓ５１２，Ｓ７０６は、形態５のリソース制限手段に対応し、ステップＳ３０８，Ｓ４０８，Ｓ５１２，Ｓ７０６は、形態１７または３２のリソース制限ステップに対応し、共通機能モジュール１２０は、形態２ないし４、１４ないし１６、２７ないし３１の第１機能モジュールに対応している。

【０１８３】

また、上記第２の実施の形態において、個別機能モジュール１３０は、形態２ないし４、１４ないし１６、２７ないし３１の第２機能モジュールに対応し、ＣＰＵ３０は、形態２６、２８または３１の演算手段に対応している。

次に、本発明の第３の実施の形態を図面を参照しながら説明する。図２５ないし図３０は、本発明に係るリソース管理プログラムおよび、並びにの実施の形態を示す図である。

【０１８４】

本実施の形態は、本発明に係るリソース管理プログラムおよび、並びにを、上記第１お

よび第２の実施の形態で使用したリソース換算テーブル２２を生成する場合について適用したものである。なお、以下、上記第１および第２の実施の形態と異なる部分についてのみ説明し、上記第１および第２の実施の形態と重複する部分については同一の符号を付して説明を省略する。

【０１８５】

まず、ネットワークプリンタの構成を詳細に説明する。

ネットワークプリンタは、ＣＰＵ、ＲＯＭ、ＲＡＭおよびＩ／Ｆ等をバス接続した一般的なコンピュータと同一機能を有して構成されている。

Ｉ／Ｆには、外部装置として、ヒューマンインターフェースとしてデータの入力および表示が可能なタッチパネル等からなる操作パネルと、データやテーブル等をファイルとして格納する記憶装置と、印刷ヘッド、ヘッド駆動部その他の印刷に必要な機構からなるプリンタエンジンと、ネットワーク１９９に接続するための信号線とが接続されている。

【０１８６】

記憶装置は、テストモジュールをネットワークプリンタで実行した実行結果を登録する実行結果登録テーブルを記憶している。

図２５は、実行結果登録テーブル５２０のデータ構造を示す図である。

実行結果登録テーブル５２０には、図２５に示すように、各テストモジュールごとに１つのレコードが登録される。各レコードは、テストモジュールの名称を登録するフィールド５２２と、テストモジュールがネットワークプリンタで使用するメモリ量を登録するフィールド５２４と、テストモジュールがネットワークプリンタで起動するクラス数を登録するフィールド５２６と、テストモジュールの処理時間を登録するフィールド５２８とを含んで登録されている。

【０１８７】

図２５の例では、第１段目のレコードには、テストモジュールの名称として「Ｔ１（メモリを使用形態Ａでのみ使用）」が、メモリ量として「100」が、クラス数として「５]が、処理時間として「30」がそれぞれ登録されている。これは、テストモジュールＴ１がメモリを使用形態Ａでのみ使用するものであり、その実行により、ネットワークプリンタでメモリ量を100[byte]使用し、かつ、クラスを30個起動することを示している。

【０１８８】

ＣＰＵは、ＲＯＭの所定領域に格納されている所定のプログラムを起動させ、そのプログラムに従って、図２６のフローチャートに示す使用リソース量測定処理を実行する。

図２６は、使用リソース量測定処理を示すフローチャートである。

使用リソース量測定処理は、ＣＰＵにおいて実行されると、図２６に示すように、まず、ステップＳ９００に移行する。

【０１８９】

ステップＳ９００では、実行すべきテストモジュールを登録したテストモジュールリストを取得し、ステップＳ９０２に移行して、取得したテストモジュールリストに基づいてテストモジュールを実行し、ステップＳ９０４に移行する。

ステップＳ９０４では、実行したテストモジュールが使用するメモリ量および起動するクラス数を測定し、ステップＳ９０６に移行して、実行したテストモジュールの処理時間を測定し、ステップＳ９０８に移行して、測定した使用メモリ量、起動クラス数および測定時間をテストモジュールの名称と対応付けて実行結果登録テーブル５２０に登録し、ステップＳ９１０に移行する。

【０１９０】

ステップＳ９１０では、取得したテストモジュールリストに基づいて、実行すべきテストモジュールが存在するか否かを判定し、実行すべきテストモジュールが存在すると判定したとき(Yes)は、ステップＳ９０２に移行するが、そうでないと判定したとき(No)は、一連の処理を終了して元の処理に復帰させる。

使用リソース量測定処理により得られた実行結果登録テーブル５２０は、ホスト端末１００でリソース換算テーブル２２を生成するのに使用される。

【０１９１】
次に、ホスト端末１００の構成を詳細に説明する。
ＣＰＵ３０は、ＲＯＭ３２の所定領域に格納されている所定のプログラムを起動させ、そのプログラムに従って、図２７のフローチャートに示すリソース換算テーブル生成処理を実行する。
図２７は、リソース換算テーブル生成処理を示すフローチャートである。
【０１９２】
リソース換算テーブル生成処理は、リソース換算テーブル２２を生成するとともにステップＳ７１３での待機時間を決定する処理であって、ＣＰＵ３０において実行されると、図２７に示すように、まず、ステップＳ１０００に移行する。
ステップＳ１０００では、テストモジュールリストを取得し、ステップＳ１００２に移行して、リソース換算テーブル２２を初期化し、ステップＳ１００４に移行して、待機時間を「０」に設定し、ステップＳ１００６に移行する。
【０１９３】
ステップＳ１００６では、取得したテストモジュールリストに基づいてすべてのテストモジュールを実行する。ここで、テストモジュールは、個別機能モジュール１３０と同様に、図１０ないし図１７、および図２４のフローチャートに示す処理により実行する。したがって、ステップＳ７１３で待機時間だけ処理を待機することにより処理時間が調整される。
【０１９４】
次いで、ステップＳ１００８に移行して、実行した各テストモジュールが使用するメモリ量および起動するクラス数を測定し、ステップＳ１０１０に移行して、実行した各テストモジュールの処理時間を測定し、ステップＳ１０１２に移行する。
ステップＳ１０１２では、各テストモジュールに対応する処理時間を実行結果登録テーブル５２０から読み出し、ステップＳ１０１４に移行して、読み出した処理時間の合計と、各テストモジュールについて測定した処理時間の合計とが一致しまたはその差分が所定閾値以下であるか否かを判定し、処理時間の合計が一致しないまたはその差分が所定閾値よりも大きいと判定したとき(No)は、ステップＳ１０１６に移行する。
【０１９５】
ステップＳ１０１６では、読み出した処理時間の合計が、各テストモジュールについて測定した処理時間の合計よりも短いか否かを判定し、読み出した処理時間の合計の方が短いと判定したとき(Yes)は、ステップＳ１０１８に移行して、待機時間を「１」減算し、ステップＳ１００６に移行する。
一方、ステップＳ１０１６で、各テストモジュールについて測定した処理時間の合計の方が短いと判定したとき(No)は、ステップＳ１０２０に移行して、待機時間を「１」加算し、ステップＳ１００６に移行する。
【０１９６】
一方、ステップＳ１０１４で、処理時間の合計が一致しまたはその差分が所定閾値以下であると判定したとき(Yes)は、ステップＳ１０２２に移行して、各テストモジュールに対応する使用メモリ量および起動クラス数を実行結果登録テーブル５２０から読み出し、ステップＳ１０２４に移行する。
ステップＳ１０２４では、各テストモジュールごとに、そのテストモジュールについて測定した使用メモリ量および起動クラス数と、読み出した使用メモリ量および起動クラス数のうちそのテストモジュールに対応するものとに基づいて換算率を算出し、ステップＳ１０２６に移行して、リソースの種別および使用形態ごとに、各テストモジュールについて算出した換算率のうち最大値をリソース換算テーブル２２に登録し、一連の処理を終了して元の処理に復帰させる。
【０１９７】
次に、本実施の形態の動作を説明する。
ネットワークプリンタでは、ステップＳ９０２～Ｓ９０８を繰り返して経て、各テスト

モジュールが実行され、実行された各テストモジュールの使用メモリ量、起動クラス数および処理時間が測定され、測定された使用メモリ量、起動クラス数および処理時間が実行結果登録テーブル５２０に登録される。

【０１９８】

次に、ネットワークプリンタで得られた実行結果登録テーブル５２０をホスト端末１００に格納し、ホスト端末１００でリソース換算テーブル生成処理を実行する。

ホスト端末１００では、ステップＳ１００６～Ｓ１０１０を経て、各テストモジュールが実行され、実行された各テストモジュールの処理時間が測定される。そして、ステップＳ１０１２，Ｓ１０１４を経て、各テストモジュールに対応する処理時間が実行結果登録テーブル５２０から読み出され、読み出された処理時間の合計と、各テストモジュールについて測定された処理時間の合計とが一致するか否かが判定される。その結果、処理時間の合計が一致しないと判定されると、読み出された処理時間の合計の方が短い場合は、ステップＳ１０１８を経て、待機時間が減算されるが、各テストモジュールについて測定された処理時間の合計の方が短い場合は、ステップＳ１０２０を経て、待機時間が加算される。ステップＳ１００６～Ｓ１０２０の処理は、処理時間の合計が一致するまで繰り返し行われる。

【０１９９】

図２８は、ネットワークプリンタで３つのテストモジュールＴ１～Ｔ３を実行した場合の実行結果を示す表である。

ネットワークプリンタでテストモジュールＴ１～Ｔ３を実行した場合、テストモジュールＴ１～Ｔ３の処理時間の合計は、図２８に示すように、例えば、「120」となった。

図２９は、待機時間を「０」に設定し、ホスト端末１００で３つのテストモジュールＴ１～Ｔ３を実行した場合の実行結果を示す表である。

【０２００】

待機時間を「０」に設定し、ホスト端末１００でテストモジュールＴ１～Ｔ３を実行した場合、テストモジュールＴ１～Ｔ３の処理時間の合計は、図２９に示すように、例えば、「80」となった。この場合、ネットワークプリンタでの処理時間の合計と、ホスト端末１００での処理時間の合計とが一致しないので、待機時間が加算されて処理時間が調整される。

【０２０１】

図３０は、待機時間を「１０」に設定し、ホスト端末１００で３つのテストモジュールＴ１～Ｔ３を実行した場合の実行結果を示す表である。

次に、待機時間を「１０」に設定し、ホスト端末１００でテストモジュールＴ１～Ｔ３を実行した場合、テストモジュールＴ１～Ｔ３の処理時間の合計は、図３０に示すように、例えば、「120」となった。この場合、ネットワークプリンタでの処理時間の合計と、ホスト端末１００での処理時間の合計とが一致するので、待機時間は、「１０」として決定される。

【０２０２】

このように待機時間が決定されると、ステップＳ１０２２，Ｓ１０２４を経て、各テストモジュールに対応する使用メモリ量および起動クラス数が実行結果登録テーブル５２０から読み出され、測定された使用メモリ量および起動クラス数、並びに読み出された使用メモリ量および起動クラス数に基づいて各テストモジュールごとに換算率が算出される。そして、ステップＳ１０２６を経て、各テストモジュールについて算出された換算率のうち最大値がリソース換算テーブル２２に登録される。

【０２０３】

このような処理により得られた待機時間およびリソース換算テーブル２２は、上記第１および第２の実施の形態で使用することができる。

このようにして、本実施の形態では、テストモジュールがネットワークプリンタで使用するリソースの量を実行結果登録テーブル５２０から読み出し、テストモジュールがホスト端末１００で使用するリソースの量を測定し、読み出したリソースの量および測定した

リソースの量に基づいて換算率を算出し、算出した換算率に基づいてリソース換算テーブル２２を生成するようになっている。

【０２０４】

これにより、ホスト端末１００およびネットワークプリンタでテストモジュールを実行させるだけでよいので、リソース換算テーブル２２を容易に生成することができる。

さらに、本実施の形態では、ネットワークプリンタにおけるテストモジュールの処理時間を実行結果登録テーブル５２０から読み出し、ホスト端末１００におけるテストモジュールの処理時間を測定し、読み出した処理時間および測定した処理時間に基づいてホスト端末１００での待機時間を算出するようになっている。

【０２０５】

これにより、個別機能モジュール１３０をホスト端末１００で実行する場合に、得られた待機時間を実行中に設ければ、ホスト端末１００およびネットワークプリンタで個別機能モジュール１３０の処理時間が等しくなるように調整することができる。

次に、本発明の第４の実施の形態を図面を参照しながら説明する。図３１は、本発明に係るリソース管理方法および、並びにの実施の形態を示す図である。

【０２０６】

本実施の形態は、本発明に係るリソース管理方法および、並びにを、上記第１および第２の実施の形態で生成したログ情報に基づいて個別機能モジュール１３０を認証する場合について適用したものである。なお、以下、上記第１および第２の実施の形態と異なる部分についてのみ説明し、上記第１および第２の実施の形態と重複する部分については同一の符号を付して説明を省略する。

【０２０７】

まず、ホスト端末１００の構成を詳細に説明する。

ＣＰＵ３０は、ＲＯＭ３２の所定領域に格納されている所定のプログラムを起動させ、そのプログラムに従って、図３１のフローチャートに示すモジュール認証処理を実行する。

図３１は、モジュール認証処理を示すフローチャートである。

【０２０８】

モジュール認証処理は、ＣＰＵ３０において実行されると、図３１に示すように、まず、ステップＳ１１００に移行する。

ステップＳ１１００では、記憶装置４２のログファイルからログ情報を読み出し、ステップＳ１１０２に移行して、読み出したログ情報に基づいて、個別機能モジュール１３０の起動クラス数または使用メモリ量が上限に達したか否かを判定し、起動クラス数または使用メモリ量が上限に達していないと判定したとき(No)は、ステップＳ１１０４に移行する。

【０２０９】

ステップＳ１１０４では、読み出したログ情報に基づいて、個別機能モジュール１３０がネットワークプリンタにインストール不可能である否かを判定し、個別機能モジュール１３０がインストール可能であると判定したとき(No)は、ステップＳ１１０６に移行する。

ステップＳ１１０６では、記憶装置４２のログファイルに未処理のログ情報が存在するか否かを判定し、未処理のログ情報が存在しないと判定したとき(No)は、ステップＳ１１０８に移行して、個別機能モジュール１３０の実行ファイルを記憶装置４２から読み出し、ステップＳ１１１０に移行する。

【０２１０】

ステップＳ１１１０では、読み出した実行ファイルに電子署名情報を付加し、ステップＳ１１１２に移行して、電子署名情報を付加した実行ファイルを記憶装置４２に保存し、一連の処理を終了して元の処理に復帰させる。

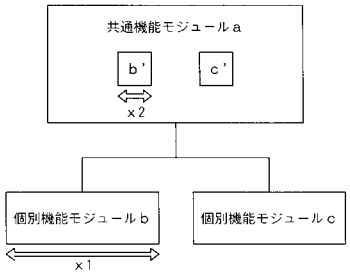
一方、ステップＳ１１０６で、未処理のログ情報が存在すると判定したとき(Yes)は、ステップＳ１１００に移行する。

【０２１１】
一方、ステップＳ１１０４で、個別機能モジュール１３０がインストール不可能であると判定したとき(Yes)、およびステップＳ１１０２で、起動クラス数または使用メモリ量が上限に達したと判定したとき(Yes)はいずれも、ステップＳ１１１４に移行して、個別機能モジュール１３０が認証不可能であることを示すメッセージを表示装置４４に表示し、一連の処理を終了して元の処理に復帰させる。
【０２１２】
次に、本実施の形態の動作を説明する。
ホスト端末１００では、上記第１および第２の実施の形態においてログファイルが生成されると、ステップＳ１１００～Ｓ１１０６を繰り返しを経て、ログファイルからログ情報が順次読み出され、個別機能モジュール１３０の起動クラス数または使用メモリ量が上限に達したか否か、および個別機能モジュール１３０がネットワークプリンタにインストール不可能であるか否かが判定される。図１９に示すログファイルのように、ログファイルに含まれるすべてのログ情報について、起動クラス数または使用メモリ量が上限に達せずかつインストール可能であると判定されると、ステップＳ１１０８～Ｓ１１１２を経て、個別機能モジュール１３０の実行ファイルが読み出され、読み出された実行ファイルに電子署名情報が付加されて保存される。
【０２１３】
これに対し、図１８に示すログファイルのように、ログファイルに含まれるいずれかのログ情報について、起動クラス数または使用メモリ量が上限に達しまたはインストール不可能であると判定されると、ステップＳ１１１４を経て、認証不可能であることを示すメッセージが表示される。
このようにして、本実施の形態では、ログファイルからログ情報を読み出し、読み出したログ情報に基づいて個別機能モジュール１３０が使用するリソースの量が上限に達したか否かを判定し、リソースの量が上限に達していないと判定したときは、個別機能モジュール１３０の実行ファイルに電子認証情報を付加するようになっている。
【０２１４】
これにより、使用するリソースの量が上限に達しない個別機能モジュール１３０に対してのみ電子署名情報が付加されるので、個別機能モジュール１３０の動作を比較的確実に保証することができる。
さらに、本実施の形態では、ログファイルからログ情報を読み出し、読み出したログ情報に基づいて個別機能モジュール１３０がネットワークプリンタにインストール不可能であるか否かを判定し、インストール可能であると判定したときは、個別機能モジュール１３０の実行ファイルに電子認証情報を付加するようになっている。
【０２１５】
これにより、ネットワークプリンタにインストール可能である個別機能モジュール１３０に対してのみ電子署名情報が付加されるので、個別機能モジュール１３０の動作をさらに確実に保証することができる。
なお、上記第１および第２の実施の形態においては、リソース換算テーブル２２は、各テストモジュールについて決定された換算率のうち最大値を登録して構成したが、これに限らず、各テストモジュールについて決定された換算率のうち平均値を登録して構成することもできる。
【０２１６】
これにより、リソース換算テーブル２２には、各テストモジュールについて決定された換算率のうち平均値が登録されているので、機能モジュールの動作をある程度の確実性をもって保証することができるとともにネットワークプリンタで使用されるリソースの量を抑制することができる。
また、上記第１および第２の実施の形態においては、個別機能モジュール１３０が使用するリソースの量が上限に達したことを示すログ情報を生成するように構成したが、これに限らず、個別機能モジュール１３０が使用するリソースの量が上限に達したことを示す

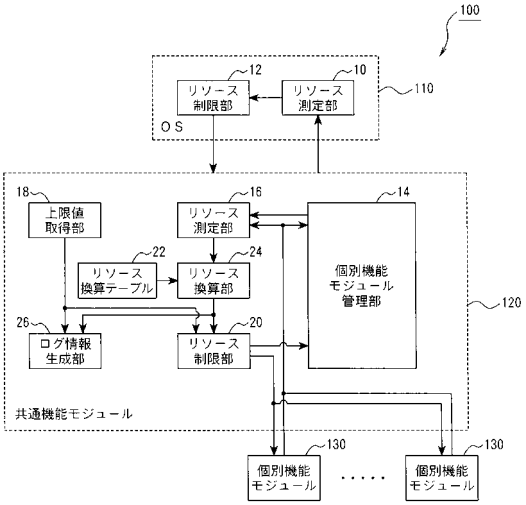
メッセージを表示装置４４に表示するように構成することもできる。
【０２１７】
また、上記第１および第２の実施の形態においては、リソース換算テーブル２２に換算率として「１」を登録した例を示したが、これに限らず、ホスト端末１００とネットワークプリンタの仕様の差異によってはそれ以外の値を登録することもできる。
また、上記第１および第２の実施の形態においては、リソースの量として使用メモリ量および起動クラス数を制限するように構成したが、これに限らず、ソケット接続数、ファイル接続数、ファイル数、ファイル容量、クラスサイズ、ＺＩＰメモリ容量、ＣＰＵ利用量、ソケット通信量およびファイル読み書き量を制限するように構成することもできる。
【０２１８】
また、上記第１ないし第４の実施の形態において、図１０ないし図１７、図２４、図２７および図３１のフローチャートに示す処理を実行するにあたってはいずれも、ＲＯＭ３２にあらかじめ格納されている制御プログラムを実行する場合について説明したが、これに限らず、これらの手順を示したプログラムが記憶された記憶媒体から、そのプログラムをＲＡＭ３４に読み込んで実行するようにしてもよい。
【０２１９】
また、上記第３の実施の形態において、図２６のフローチャートに示す処理を実行するにあたってはいずれも、ネットワークプリンタのＲＯＭにあらかじめ格納されている制御プログラムを実行する場合について説明したが、これに限らず、これらの手順を示したプログラムが記憶された記憶媒体から、そのプログラムをネットワークプリンタのＲＡＭに読み込んで実行するようにしてもよい。
【０２２０】
ここで、記憶媒体とは、ＲＡＭ、ＲＯＭ等の半導体記憶媒体、ＦＤ、ＨＤ等の磁気記憶型記憶媒体、ＣＤ、ＣＤＶ、ＬＤ、ＤＶＤ等の光学的読取方式記憶媒体、ＭＯ等の磁気記憶型／光学的読取方式記憶媒体であって、電子的、磁気的、光学的等の読み取り方法のいかんにかかわらず、コンピュータで読み取り可能な記憶媒体であれば、あらゆる記憶媒体を含むものである。
【０２２１】
また、上記第１ないし第４の実施の形態においては、本発明に係るリソース管理システムおよびリソース管理プログラム、並びにリソース管理方法を、ホスト端末１００上のＪＡＶＡ（登録商標）アプリケーションの実行環境において、ネットワークプリンタの動作を制御するためのＪＡＶＡ（登録商標）クラスセットをエミュレーションする場合について適用したが、これに限らず、本発明の主旨を逸脱しない範囲で他の場合にも適用可能である。ネットワークプリンタに代えて、例えば、プロジェクタ、電子ペーパ、ホームゲートウェイ、パソコン、ＰＤＡ（Personal Digital Assistant）、ネットワークストレージ、オーディオ機器、携帯電話、ＰＨＳ（登録商標）（Personal Handyphone System）、ウォッチ型ＰＤＡ、ＳＴＢ（Set Top Box）、ＰＯＳ（Point Of Sale）端末、ＦＡＸ機、電話（ＩＰ電話等も含む。）、その他のデバイスに適用することができる。
【図面の簡単な説明】
【０２２２】
【図１】ＪＡＶＡ（登録商標）ソフトウェアの構成を示す図である。
【図２】ホスト端末１００の機能概要を示す機能ブロック図である。
【図３】ホスト端末１００のハードウェア構成を示すブロック図である。
【図４】リソース制限情報４００のデータ構造を示す図である。
【図５】モジュール情報４２０のデータ構造を示す図である。
【図６】実行環境情報登録テーブル４４０のデータ構造を示す図である。
【図７】リソース換算テーブル２２のデータ構造を示す図である。
【図８】リソース管理テーブル４６０のデータ構造を示す図である。
【図９】イベントリスナテーブル４８０のデータ構造を示す図である。
【図１０】個別機能モジュール制御処理を示すフローチャートである。

【図１１】実行可否判定処理を示すフローチャートである。
【図１２】モジュール起動処理を示すフローチャートである。
【図１３】イベントリスナ登録処理を示すフローチャートである。
【図１４】クラス読込処理を示すフローチャートである。
【図１５】イベントリスナ制御処理を示すフローチャートである。
【図１６】イベントリスナ実行処理を示すフローチャートである。
【図１７】インスタンス削除処理を示すフローチャートである。
【図１８】エラーが発生した場合のログファイルの内容を示す図である。
【図１９】エラーが発生しない場合のログファイルの内容を示す図である。
【図２０】リソース管理対象となる個別機能モジュールｂ，ｃを並列に実行した場合を示すタイムチャートである。
【図２１】ホスト端末１００の機能概要を示す機能ブロック図である。
【図２２】リソース換算テーブル２２のデータ構造を示す図である。
【図２３】リソース管理テーブル４６０のデータ構造を示す図である。
【図２４】実行可否判定処理を示すフローチャートである。
【図２５】実行結果登録テーブル５２０のデータ構造を示す図である。
【図２６】使用リソース量測定処理を示すフローチャートである。
【図２７】リソース換算テーブル生成処理を示すフローチャートである。
【図２８】ネットワークプリンタで３つのテストモジュールＴ１～Ｔ３を実行した場合の実行結果を示す表である。
【図２９】待機時間を「０」に設定し、ホスト端末１００で３つのテストモジュールＴ１～Ｔ３を実行した場合の実行結果を示す表である。
【図３０】待機時間を「１０」に設定し、ホスト端末１００で３つのテストモジュールＴ１～Ｔ３を実行した場合の実行結果を示す表である。
【図３１】モジュール認証処理を示すフローチャートである。
【符号の説明】
【０２２３】
１００…ホスト端末，　１１０…ＯＳ，　１２０…共通機能モジュール，　１３０…個別機能モジュール，　１０，１６…リソース測定部，　１２，２０…リソース制限部，　１４…個別機能モジュール管理部，　１８…上限値取得部，　２２…リソース換算テーブル，　２４，２８…リソース換算部，　２６…ログ情報生成部，　３０…ＣＰＵ，　３２…ＲＯＭ，　３４…ＲＡＭ，　３８…Ｉ／Ｆ，　４０…入力装置，　４２…記憶装置，　４４…表示装置，　４００…リソース制限情報，　４２０…モジュール情報，　４４０…実行環境情報登録テーブル，　４６０…リソース管理テーブル，　４８０…イベントリスナテーブル，　５２０…実行結果登録テーブル，　１９９…ネットワーク

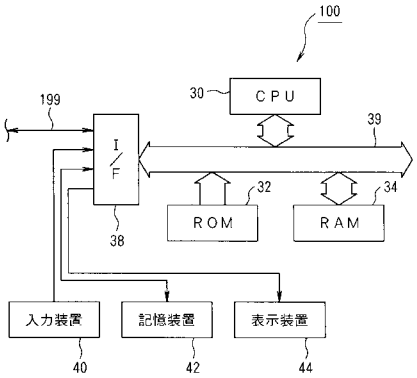
【図１】



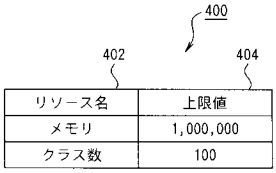
【図２】



【図３】

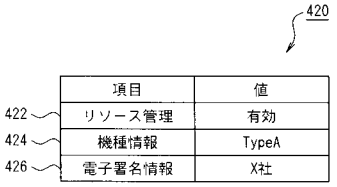


【図４】



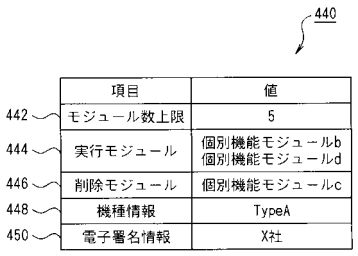
リソース名	上限値
メモリ	1,000,000
クラス数	100

【図５】



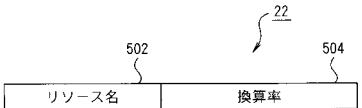
項目	値
リソース管理	有効
機種情報	TypeA
電子署名情報	X社

【図６】



項目	値
モジュール数上限	5
実行モジュール	個別機能モジュールb 個別機能モジュールd
削除モジュール	個別機能モジュールc
機種情報	TypeA
電子署名情報	X社

【図７】



リソース名	換算率
メモリ 使用形態A	1
メモリ 使用形態B	1.5
メモリ 使用形態C	2
クラス数	1

【図８】

460

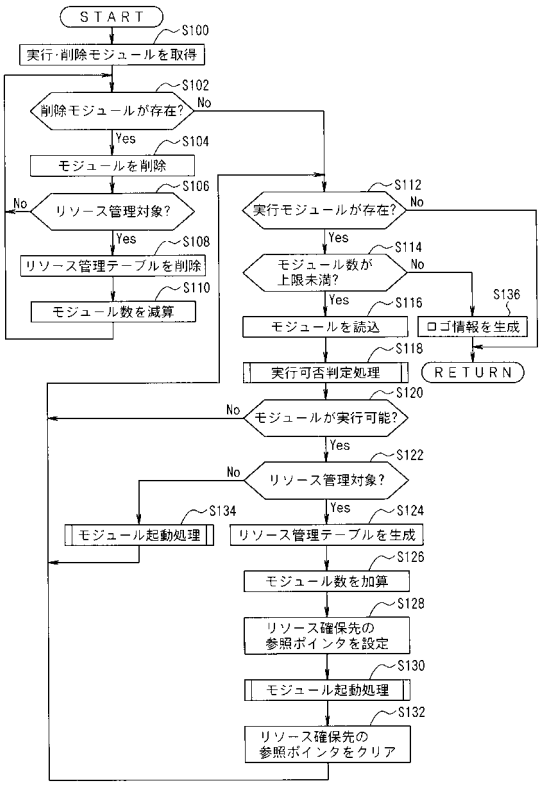
462 リソース名	464 上限値	466 現在値	468 換算値
メモリ	1,000,000	メモリ使用形態A 200,000 メモリ使用形態B 100,000 メモリ使用形態C 150,000	650,000
クラス数	100	20	20

【図９】

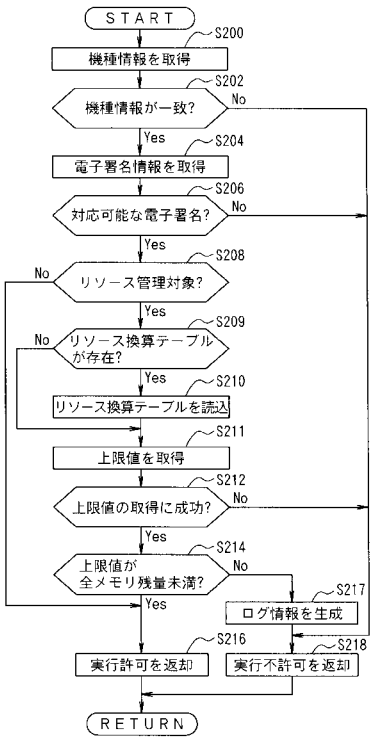
480

482 イベントリスナ名
個別機能モジュールbイベントリスナ
個別機能モジュールcイベントリスナ

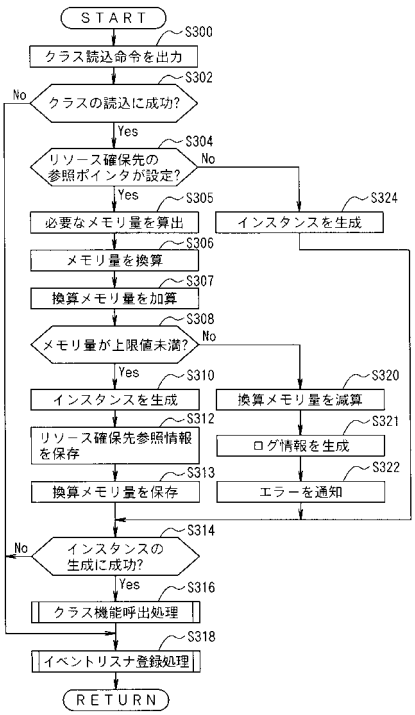
【図１０】



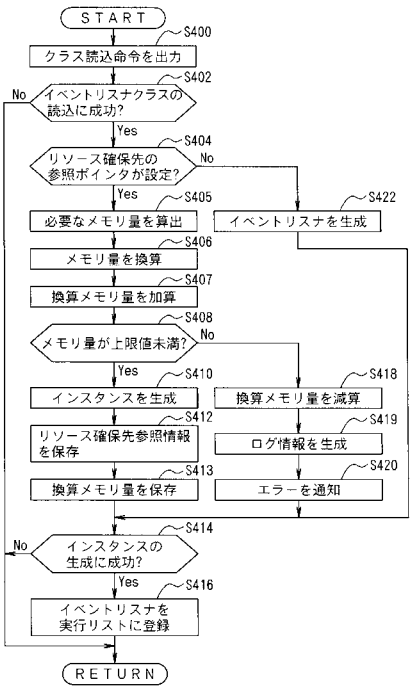
【図１１】



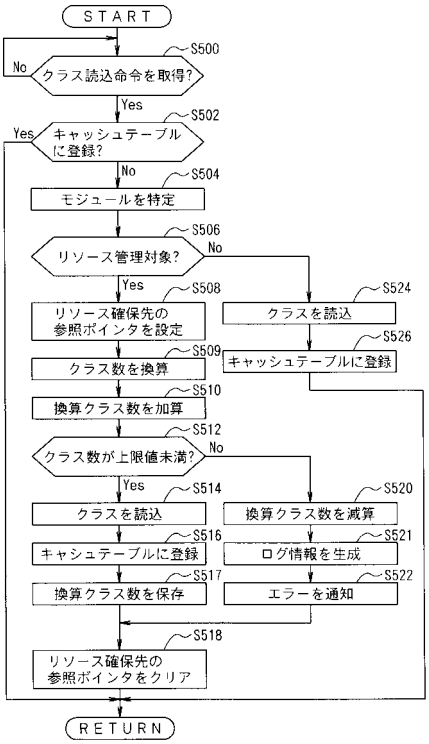
【図１２】



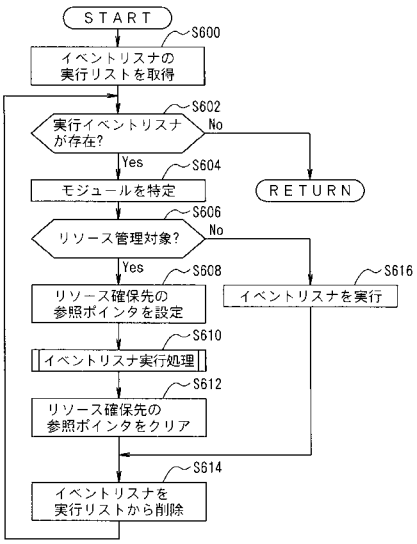
【図１３】



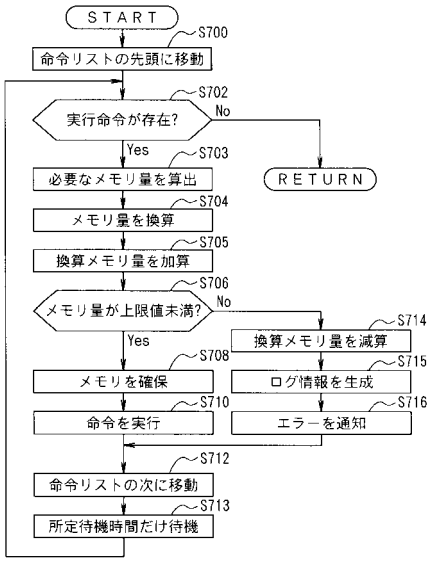
【図１４】



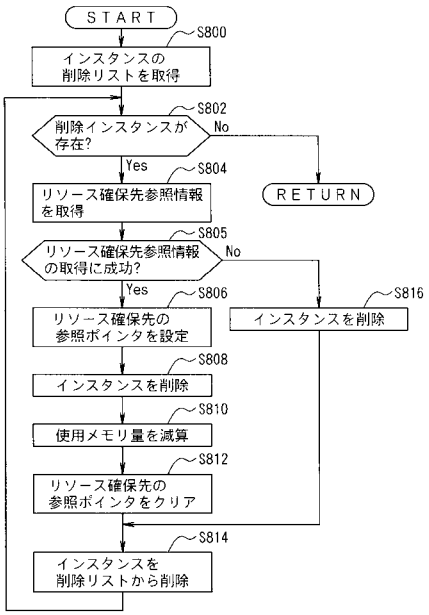
【図１５】



【図１６】



【図１７】



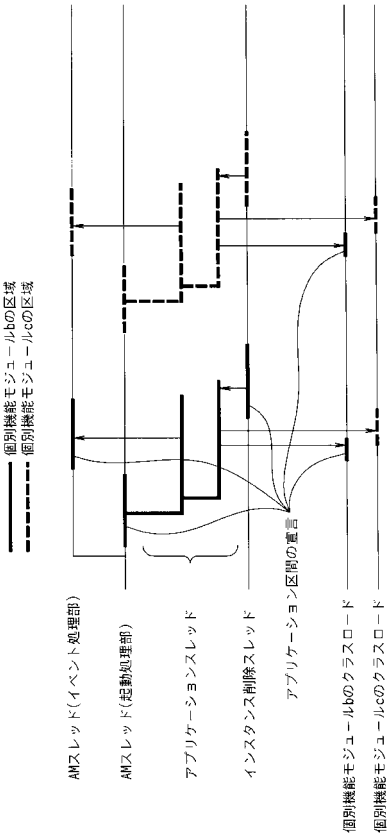
【図１８】

時刻	ログ内容
2004/10/10 09:00:00	個別機能モジュールを起動
2004/10/10 09:01:00	メモリ量が上限に達しました。(上限値1,000,000<使用量1,000,200)
2004/10/10 09:05:00	クラス数が上限に達しました。(上限値100<使用量101)
2004/10/10 09:06:00	クラス数が上限に達しました。(上限値100<使用量101)
2004/10/10 09:10:00	個別機能モジュールを停止

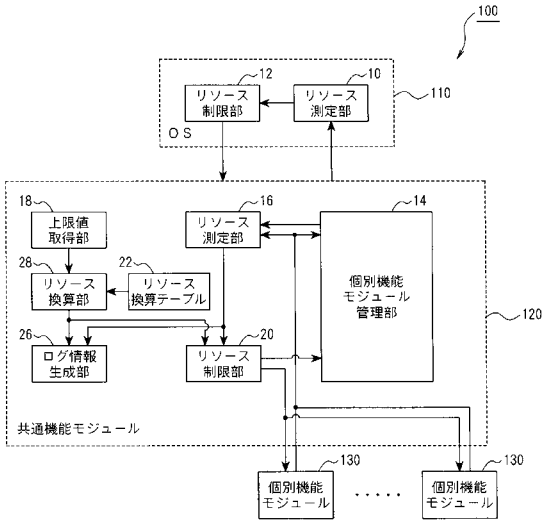
【図１９】

時刻	ログ内容
2004/10/10 09:00:00	個別機能モジュールを起動
2004/10/10 09:10:00	個別機能モジュールを停止

【図２０】



【図２１】



【図２２】

22

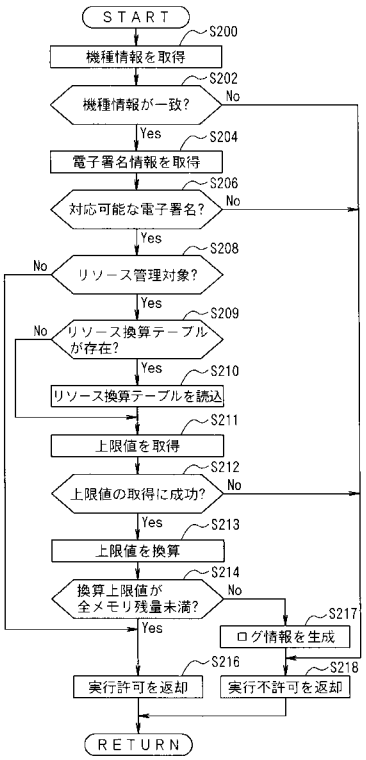
502 リソース名	504 換算率
メモリ	1.5
クラス数	1

【図２３】

460

462 リソース名	464 上限値	470 換算値	466 現在値
メモリ	1,000,000	666,666	450,000 （内訳 メモリ使用形態A 200,000 メモリ使用形態B 100,000 メモリ使用形態C 150,000）
クラス数	100	20	20

【図２４】

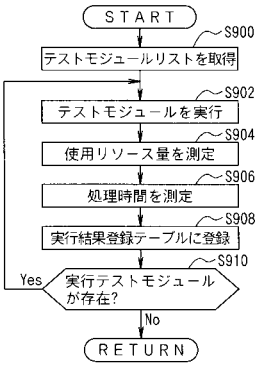


【図２５】

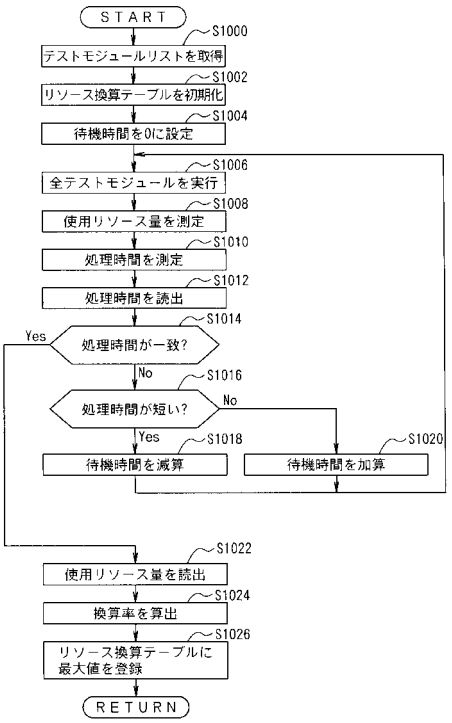
520

522 テストモジュール名	524 メモリ量	526 クラス数	528 処理時間
T1(メモリを使用形態Aでのみ使用)	100	5	30
T2(メモリを使用形態Bでのみ使用)	75	10	40
T3(メモリを使用形態Cでのみ使用)	150	2	50

【図２６】



【図２７】



【図２８】

テストモジュール名	メモリ量	クラス数	処理時間	処理時間合計
T1(メモリを使用形態Aでのみ使用)	100	5	30	120
T2(メモリを使用形態Bでのみ使用)	75	10	40	
T3(メモリを使用形態Cでのみ使用)	150	2	50	

【図２９】

待機時間
0

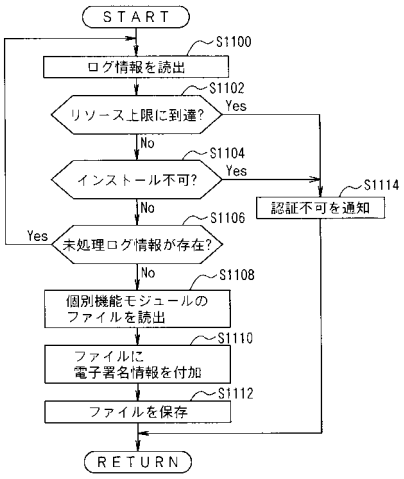
テストモジュール名	メモリ量	クラス数	処理時間	処理時間合計
T1(メモリを使用形態Aでのみ使用)	100	5	10	80
T2(メモリを使用形態Bでのみ使用)	50	10	20	
T3(メモリを使用形態Cでのみ使用)	75	2	50	

【図３０】

待機時間
10

テストモジュール名	メモリ量	クラス数	処理時間	処理時間合計
T1(メモリを使用形態Aでのみ使用)	100	5	30	120
T2(メモリを使用形態Bでのみ使用)	50	10	35	
T3(メモリを使用形態Cでのみ使用)	75	2	55	

【図３１】



【手続補正書】
【提出日】平成17年11月25日(2005.11.25)
【手続補正１】
【補正対象書類名】明細書
【補正対象項目名】０１３６
【補正方法】変更
【補正の内容】
【０１３６】
イベントリスナ実行処理は、ステップＳ６１０において実行されると、図１６に示すように、まず、ステップＳ７００に移行する。
ステップＳ７００では、イベントリスナに含まれる命令リストの先頭にプログラムポインタを移動し、ステップＳ７０２に移行して、プログラムポインタが指し示すアドレスに実行すべき命令が存在するか否かを判定し、実行すべき命令が存在すると判定したとき(Yes)は、ステップＳ７０３に移行して、命令の実行に必要なメモリ量を算出し、ステップＳ７０４に移行する。

フロントページの続き

Ｆターム(参考) 5B042 GA36 HH20 JJ29 KK13 MA10 MC29