(19)日本国特許庁(JP)

(12)特許公報(B2)

(11)特許番号
特許第7013165号
(P7013165)

(45)発行日　令和4年1月31日(2022.1.31)

(24)登録日　令和4年1月21日(2022.1.21)

(51)国際特許分類

Ｈ０４Ｌ　67/00　(2022.01)		Ｈ０４Ｌ	67/00	
Ｇ０６Ｆ　11/34　(2006.01)		Ｇ０６Ｆ	11/34	１７６
Ｇ０６Ｆ　11/30　(2006.01)		Ｇ０６Ｆ	11/30	１５５
		Ｇ０６Ｆ	11/30	１４０Ａ

ＦＩ

請求項の数　8　（全27頁）

(21)出願番号　特願2017-153023(P2017-153023)
(22)出願日　平成29年8月8日(2017.8.8)
(65)公開番号　特開2019-32686(P2019-32686A)
(43)公開日　平成31年2月28日(2019.2.28)
審査請求日　令和2年7月17日(2020.7.17)

(73)特許権者　000001007
キヤノン株式会社
東京都大田区下丸子３丁目３０番２号
(74)代理人　110002767
特許業務法人ひのき国際特許事務所
(74)代理人　100199820
弁理士　西脇 博志
(74)代理人　100145827
弁理士　水垣 親房
(72)発明者　中澤 紀之
東京都大田区下丸子３丁目３０番２号
キヤノン株式会社内
審査官　中川 幸洋

最終頁に続く

(54)【発明の名称】　管理装置、管理装置の制御方法、及びプログラム

(57)【特許請求の範囲】
【請求項１】
複数のデバイスを監視可能なエージェントが動作する１又は複数の監視装置から送信される情報に基づいてデバイスを管理する管理装置であって、
１以上の指定デバイスに対して実行すべき処理の対象となるデバイスのデバイス識別子が指定されたタスク情報を生成する第１生成手段と、
前記タスク情報に基づいて、前記処理の対象となる一部のデバイスのデバイス識別子及び該一部のデバイスに対応するエージェントのエージェント識別子が指定されたサブタスク情報を複数生成する第２生成手段と、
前記タスク情報及び前記サブタスク情報を管理する管理手段と、
複数のエージェントにタスクの開始を通知する通知手段と、
前記エージェント識別子が指定されたサブタスク情報を該エージェント識別子に対応するエージェントに送信する送信手段と、
前記エージェントから受信したサブタスクの実行結果を、該サブタスクに対応するサブタスク情報に設定する第１設定手段と、
前記生成された全てのサブタスク情報に実行結果が設定されたことに応じて、これらのサブタスク情報に対応するタスク情報に終了日時を設定する第２設定手段と、を有し、
前記第１生成手段は、さらに、デバイスを探索する処理のためのタスク情報を生成でき、
前記第２生成手段は、前記デバイスを探索する処理のためのタスク情報について、対象となるデバイスのデバイス識別子が指定されず、対象となるエージェントのエージェント

識別子及び探索の対象となるアドレスが指定された探索処理のためのサブタスク情報を生成し、
　前記第１設定手段は、前記エージェントから前記探索処理のためのサブタスク情報で指定された前記探索の対象となる全てのアドレスに対応する実行結果を受信したことに応じて、前記サブタスク情報に実行結果を設定する
ことを特徴とする管理装置。
【請求項２】
前記第１生成手段は、前記タスク情報を生成した際に、前記タスク情報に開始日時を設定する
ことを特徴とする請求項１に記載の管理装置。
【請求項３】
前記送信手段は、前記エージェントからの要求に応じて、該エージェントのエージェント識別子が指定されたサブタスク情報を該エージェントに送信する
ことを特徴とする請求項１又は２に記載の管理装置。
【請求項４】
前記通知手段は、ＨＴＴＰ又はＨＴＴＰＳを用いて複数のエージェントにタスクの開始を通知し、
前記第１設定手段は、ＨＴＴＰ又はＨＴＴＰＳを用いて前記エージェントからサブタスクの実行結果を受信する
ことを特徴とする請求項１～３のいずれか１項に記載の管理装置。
【請求項５】
前記１以上の指定デバイスに対して実行すべき処理の実行タイミングを含む処理の予約を受け付ける受け付け手段を有し、
前記第１生成手段は、前記処理の予約に基づいて、前記タスク情報を生成する
ことを特徴とする請求項１～４のいずれか１項に記載の管理装置。
【請求項６】
前記１以上の指定デバイスに対して実行すべき処理は、前記デバイスから情報を取得する処理、前記デバイスに設定情報を配信する処理、又は、前記デバイスにアプリケーションソフトウェアを配信する処理を含む
ことを特徴とする請求項１～５のいずれか１項に記載の管理装置。
【請求項７】
複数のデバイスを監視可能なエージェントが動作する１又は複数の監視装置から送信される情報に基づいてデバイスを管理するための制御方法であって、
１以上の指定デバイスに対して実行すべき処理の対象となるデバイスのデバイス識別子が指定されたタスク情報を生成する第１生成ステップと、
前記タスク情報に基づいて、前記処理の対象となる一部のデバイスのデバイス識別子及び該一部のデバイスに対応するエージェントのエージェント識別子が指定されたサブタスク情報を複数生成する第２生成ステップと、
複数のエージェントにタスクの開始を通知する通知ステップと、
前記エージェント識別子が指定されたサブタスク情報を該エージェント識別子に対応するエージェントに送信する送信ステップと、
前記エージェントから受信したサブタスクの実行結果を、該サブタスクに対応するサブタスク情報に設定する第１設定ステップと、
前記生成された全てのサブタスク情報に実行結果が設定されたことに応じて、これらのサブタスク情報に対応するタスク情報に終了日時を設定する第２設定ステップと、を有し、
前記第１生成ステップでは、さらに、デバイスを探索する処理のためのタスク情報が生成でき、
　前記第２生成ステップでは、前記デバイスを探索する処理のためのタスク情報について、対象となるデバイスのデバイス識別子が指定されず、対象となるエージェントのエージェント識別子及び探索の対象となるアドレスが指定された探索処理のためのサブタスク情

報が生成され、
前記第１設定ステップでは、前記エージェントから前記探索処理のためのサブタスク情報で指定された前記探索の対象となる全てのアドレスに対応する実行結果を受信したことに応じて、前記サブタスク情報に実行結果が設定される
ことを特徴とする制御方法。
【請求項８】
コンピュータを、請求項１～６のいずれか１項に記載の手段として機能させるためのプログラム。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、複数のデバイスを監視可能なエージェントが動作する１又は複数の監視装置と、前記１又は複数の監視装置から送信される情報に基づいてデバイスを管理する管理装置、管理装置の制御方法、及びプログラムに関する。
【背景技術】
【０００２】
特許文献１には、ネットワーク接続された機器を分散管理するシステムが開示されている。特許文献１では、管理を行う管理サーバを複数配置し、１つを親管理サーバ、残りを子管理サーバとし、親管理サーバは子管理サーバから各子管理サーバが管理しているネットワーク機器の各種情報を取得し、これを表示する。
【先行技術文献】
【特許文献】
【０００３】
【文献】特開２００９－１９９４５８号公報
【発明の概要】
【発明が解決しようとする課題】
【０００４】
しかし、従来の技術では、親管理サーバは子管理サーバが収集したネットワーク機器に関する情報を収集、表示するのみである。従来の技術では、親管理サーバが子管理サーバに対して管理対象のネットワーク機器への処理を指示ができないという課題があった。
【０００５】
本発明は、上記の課題を解決するためになされたものである。本発明の目的は、分散管理する複数のデバイスに処理を実行させる場合でも、処理の内容に関係なく管理装置からエージェントに対してデバイスへの処理を指示でき、効率的にデバイスに対して処理を実行可能にする仕組みを提供することである。
【課題を解決するための手段】
【０００６】
本発明は、複数のデバイスを監視可能なエージェントが動作する１又は複数の監視装置から送信される情報に基づいてデバイスを管理する管理装置であって、１以上の指定デバイスに対して実行すべき処理の対象となるデバイスのデバイス識別子が指定されたタスク情報を生成する第１生成手段と、前記タスク情報に基づいて、前記処理の対象となる一部のデバイスのデバイス識別子及び該一部のデバイスに対応するエージェントのエージェント識別子が指定されたサブタスク情報を複数生成する第２生成手段と、前記タスク情報及び前記サブタスク情報を管理する管理手段と、複数のエージェントにタスクの開始を通知する通知手段と、前記エージェント識別子が指定されたサブタスク情報を該エージェント識別子に対応するエージェントに送信する送信手段と、前記エージェントから受信したサブタスクの実行結果を、該サブタスクに対応するサブタスク情報に設定する第１設定手段と、前記生成された全てのサブタスク情報に実行結果が設定されたことに応じて、これらのサブタスク情報に対応するタスク情報に終了日時を設定する第２設定手段と、を有し、前記第１生成手段は、さらに、デバイスを探索する処理のために、探索を実行するエージェ

ントのエージェント識別子が指定されたタスク情報を生成でき、前記第２生成手段は、前記デバイスを探索する処理のためのタスク情報について、対象となるデバイスのデバイス識別子が指定されず、対象となるエージェントのエージェント識別子及び探索の対象となるアドレスが指定された探索処理のためのサブタスク情報を生成し、前記第１設定手段は、前記エージェントから前記サブタスク情報で指定された前記探索の対象となる全てのアドレスに対応する実行結果を受信したことに応じて、前記サブタスク情報に実行結果を設定することを特徴とする。

【発明の効果】

【０００７】

本発明によれば、複数のエージェントを用いて分散管理する複数のデバイスに処理を実行させる場合でも、処理の内容に関係なく管理装置からエージェントに対してデバイスへの処理を指示でき、効率的にデバイスに対して処理を実行させることができる。

【図面の簡単な説明】

【０００８】

【図１】本実施例の分散ネットワーク機器管理システムを例示する図
【図２】ホストコンピュータのハードウェア構成の一例を示すブロック図
【図３】本実施例の管理アプリケーションの構成の一例を示す図
【図４】本実施例の設定値の取得の予約タスクを作成する画面を例示する図
【図５】本実施例のタイマー処理、予約開始処理の一例を示すフローチャート
【図６】本実施例のエージェントの構成を例示する図
【図７】実施例１のタスク起動処理、タスク開始処理を例示するフローチャート
【図８】本実施例のサブタスク実行処理を例示するフローチャート
【図９】本実施例のサブタスク実行結果受信処理を例示するフローチャート
【図１０】実施例４のタスク起動処理、タスク再起動処理を例示するフローチャート
【図１１】本実施例のタスクの開始処理から結果送信までの流れを示すシーケンス図

【発明を実施するための形態】

【０００９】

以下、本発明を実施するための形態について図面を用いて説明する。

【実施例１】

【００１０】

図１は、本発明の一実施例を示す分散ネットワーク機器管理システムの構成を例示する図である。

図１において、管理装置101は、ネットワーク機器の管理を集中的に行う管理アプリケーション101aが動作するサーバ装置である。ＤＢ112は、管理アプリケーション101aにより管理されるデータを格納するデータベースである。なお、ＤＢ112は、管理装置101内に実装される構成でも、管理装置101と通信可能な他の装置に実装される構成でもよい。また、管理装置101は、１台の装置で構成されていてもよいし、複数の装置で構成されてもよく、クラウドサーバ等で構成されていてもよい。例えば、管理装置101は、パブリッククラウドサービス等を利用して構築することも可能である。

【００１１】

ネットワーク機器107、108、109、110、111は、管理アプリケーション101aにより管理されるネットワーク機器である。ネットワーク機器の一例としては、複合機やプリンタ等の画像形成装置、ネットワークカメラ、パーソナルコンピュータ、ネットワークに接続された各種の家電、電子制御機器を複数搭載する車両などが挙げられる。各種の家電には、テレビジョン、冷蔵庫、エアーコンディショナー等が含まれる。なお、ネットワーク機器を、「ネットワークデバイス」又は単位「デバイス」ともいう。

【００１２】

監視装置103、104は、ネットワーク機器を監視するエージェントアプリケーション（以下「エージェント」）103a、104aが動作するサーバ装置であり、１台でも３台以上あってもよい。なお、１つのエージェントで、複数のネットワーク機器を監視可能である。ネ

ットワーク管理者は、監視するデバイスの台数や、デバイスが設置されているネットワークセグメントを考慮して、1以上のエージェントを配置する。図１の例では、監視装置103はネットワーク機器107、108を監視し、監視装置104はネットワーク機器109、110、111を監視する。

【００１３】

なお、１つのエージェントと管理アプリケーション101aは、同じサーバ装置上に実装することも可能である。また、エージェントのインストール時に、管理アプリケーション101aのＵＲＬ、認証に必要な情報などをエージェントに対して設定可能である。

【００１４】

ネットワーク102は、管理装置101と各監視装置103、104を接続するネットワークである。また、ネットワーク105、106は、それぞれ監視装置103、104とネットワーク機器107、108と109、110、111とを接続するネットワークである。ここでネットワーク102、105、106は、それぞれ独立したネットワークでも、同一のネットワークでも構わない。

【００１５】

各エージェント103a、104aは、管理アプリケーション101aと連携し、管理アプリケーション101aからの指示に従い、ネットワーク105や106を介してネットワーク機器107～111を監視（管理）する。また、各エージェント103a、104aは、監視（管理）結果を管理アプリケーション101aに送信する。

【００１６】

管理アプリケーション101aは、各エージェント103a、104aから受信した監視（管理）結果を受信し、これを必要に応じてＤＢ112に格納する。管理アプリケーション101aは、各エージェントと各ネットワーク機器との紐づけを管理する。図１の例では、エージェント103aはネットワーク機器107、108に、エージェント104aはネットワーク機器109、110、111にそれぞれ紐づけられている。このように、管理アプリケーション101aは、監視対象全てのデバイス情報や稼働情報、エージェントの識別情報と接続情報などを一元管理する。詳細は後述するが、エージェントは、管理アプリケーション101aから取得（受信）したタスクに従いデバイスの監視処理（例えば情報取得、設定配信、アプリケーション配信など）を行う。

【００１７】

なお、各監視装置103、104、各エージェント103a、104aは、それぞれ同一の構成である。よって、以下、監視装置103、エージェント103aについての説明は、監視装置104、エージェント104aについても同様である。

【００１８】

図２は、管理アプリケーション101aやエージェント103a、104aやＤＢ112が動作するホストコンピュータのハードウェア構成の一例を示すブロック図である。

図２に示すように、管理アプリケーション101aやエージェント103a、104aやＤＢ112が動作するホストコンピュータは、ＣＰＵ201、ＲＡＭ202、ＲＯＭ203、外部記憶装置207を備える情報処理装置である。

【００１９】

ＣＰＵ201は、ＲＯＭ203や外部記憶装置207に記憶された、或いはネットワーク210よりダウンロードしたソフトウェアを実行し、システムバス209に接続された各デバイスを総括的に制御する。ＲＡＭ202は、ＣＰＵ201の主メモリあるいはワークエリアなどとして機能する。

【００２０】

外部記憶装置207はハードディスク（ＨＤ）やソリッドステートドライブ（ＳＳＤ）等からなる。外部記憶装置207は、ブートプログラム、オペレーティングシステム、認証サーバ、認証クライアント等を含む各種のアプリケーション、データベースデータ、ユーザファイル等を記憶する。

【００２１】

キーボードコントローラ（ＫＢＤＣ）204は、キーボードやポインティングデバイスからの入力情報をＣＰＵ201に送る。ビデオコントローラ（ＶＣ）205は、ＬＣＤ等からなる表示装置の表示を制御する。ディスクコントローラ（ＤＣ）206は、外部記憶装置207とのアクセスを制御する。通信コントローラ（ＮＩＣ）208は、これを介してネットワーク210に接続する。

【００２２】

表１に、ＤＢ112に格納された管理アプリケーション101aにより管理されるエージェント情報の一例を示す。

【００２３】

【表１】

エージェントID	e887bddb-f3a9-404a-a7b6-e27576d0975b
エージェント名	東京オフィス
IPアドレス	172.16.10.17
ﾃﾞﾊﾞｲｽｱﾄﾞﾚｽ	172.16.0.0-172.16.255.255

【００２４】

表１において、エージェントＩＤは、各エージェントを一意に識別するために管理アプリケーション101aが付与した識別子であり、ここではＵＵＩＤを使用している。エージェント名は、各エージェントを識別するためにユーザが付けた任意の文字列である。ＩＰアドレスは、エージェントのＩＰアドレスである。デバイスアドレスは、該エージェントに紐づくネットワーク機器のアドレスを指定する。表１の例では、172.16.0.0から172.16.255.255までのＩＰアドレスを持つネットワーク機器が指定されている。なお、デバイスアドレスは、複数の開始アドレスと終了アドレスをハイフンで区切った範囲（例えば、172.20.0.0-172.20.0.255）またはＩＰアドレス（例えば、172.20.1.10）をカンマで区切った文字列等で表すものとする。

【００２５】

表２に、ＤＢ112に格納された管理アプリケーション101aにより管理される管理対象のネットワーク機器情報の一例を示す。

【００２６】

【表２】

デバイスID	41124cd1-74f8-4191-8c39-f8027c9c05a2
エージェントID	e887bddb-f3a9-404a-a7b6-e27576d0975b
デバイス名	人事部のカラーMFP
製品名	MFP－8200
IPアドレス	172.16.10.124
MACアドレス	F4:81:39:C4:CA:F7

【００２７】

表２において、デバイスＩＤは、各ネットワーク機器を一意に識別するために管理アプリケーション101aが付与した識別子であり、ここではＵＵＩＤを使用している。エージェントＩＤは、このネットワーク機器に紐づいたエージェントのエージェントＩＤである。各ネットワーク機器は、エージェントのデバイスアドレスに基づいてエージェントに紐づいている。ここでは、表１に示したエージェントが紐づいていることを示している。

デバイス名は、各ネットワーク機器を識別するためにユーザが付けた任意の文字列である。製品名、ＩＰアドレス、ＭＡＣアドレスは、それぞれ該ネットワーク機器の製品名、ＩＰアドレス、ＭＡＣアドレスである。

【００２８】

管理アプリケーション101aが複数のネットワーク機器に対して実行する処理を、総称して「タスク」と呼ぶ。また、タスクを処理するに際して、各エージェントが各ネットワーク機器に対して実行する処理を「サブタスク」と呼ぶ。即ち、タスクは、複数のサブタスクより構成される。

【００２９】

タスクの例としては、ネットワーク機器の設定値の一覧の取得がある。タスクの別の例としては、ネットワーク機器が画像形成装置である場合、出荷以降に印刷された印刷枚数の取得がある。ここで、印刷枚数には、例えば、総印刷枚数、カラー印刷枚数、モノクロ印刷枚数、両面印刷枚数、片面印刷枚数、両面カラー印刷枚数、両面モノクロ印刷枚数等の各種印刷枚数が含まれる。また、タスクの別の例として、ネットワーク機器の状態の取得がある。ネットワーク機器が画像形成装置の場合、状態には、エラー状態、トナーや感光体などの消耗品情報、給紙部の用紙サイズや用紙残量などの入力情報が含まれる。

【００３０】

図３は、管理アプリケーション101aの構成の一例を示す図である。

図３において、ＷＥＢサーバ301は、管理アプリケーション101aが動作する管理装置101上で動作する。管理アプリケーション101aは、ＷＥＢアプリケーション302と、スケジューラ303を有する。

【００３１】

スケジューラ303は、定期的にＤＢ112を走査し、実行予定時刻を経過した予約タスクの実行の準備を行い、エージェント103aに対して、タスクの実行を指示する。ここで予約タスクとは、実行するタスクの詳細に実行スケジュールを付加したものである。実行スケジュールは、該タスクを実行する日時や、定期的な実行計画（例えば毎週月曜日の８時等）である。

【００３２】

ＷＥＢアプリケーション302は、ホストコンピュータ304で動作するＷＥＢブラウザ305経由でユーザからの要求を受信して応答を返すことによりユーザインターフェイスを提供する。或いは、ＷＥＢアプリケーション302は、エージェント103からの各種データの送受信要求を処理する。ここでは、管理アプリケーション101aのユーザインターフェイスは、ＷＥＢブラウザ305を用いて提供しているが、オペレーティングシステムの画面を用いたアプリケーションであっても構わない。

【００３３】

以下、上述のようにネットワーク接続された機器の分散管理システムにおけるタスク実行方法について説明する。本実施例では、設定値の取得タスクを例にして処理の詳細を説明する。

【００３４】

図４は、設定値の取得の予約タスクを作成するためにＷＥＢブラウザ305に表示される画面の一例を示す図である。

図４において、タスク名401は、ユーザが各予約タスクを識別するための名前を入力するためのテキストボックスである。取得データ407は、このタスクでネットワーク機器から取得する設定値を選択するためのチェックボックスである。

【００３５】

402～404は、タスクの実行スケジュール（実行タイミング）を設定するＵＩ部品である。図４の例では、毎週月曜日と金曜日の8:00にタスクを実行することを示している。ＵＩ部品402～404では、例えば、毎日午後３時、特定の日時、毎月１日の午前１０時等のスケジュールを設定することが可能である。なお、ＷＥＢアプリケーション302は、実行周期選択用ＵＩ部品402の選択に従い、403、404のＵＩ部品を動的に変更する。

【００３６】
405は、設定値の収集の対象となるネットワーク機器を選択するためのチェックボックス付きのテーブルである。406は「予約」ボタンである。ＷＥＢアプリケーション302は、ユーザによる「予約」ボタン406の押下を検出すると、ＤＢ112に予約タスクのデータを保存する。
【００３７】
表３に、ＤＢ112に保存された予約タスクのデータの一例を示す。
【００３８】
【表３】

予約タスクID	5e16e760-f275-4a82-bcd4-a9ee273af6f9
タスククラス	GetSettings
タスク名	設定値の定期取得
タスク詳細	["network", "security"]
ｽｹｼﾞｭｰﾙ設定	{"recurs": "weekly", "dow": [...], ... }
開始日時	2017-04-29 16:23:00Z

【００３９】
表３において、予約タスクＩＤは、各予約タスクを一意に識別するために管理アプリケーション101aが付与した識別子であり、ここではＵＵＩＤを使用している。タスククラスは、予約タスクの作成、編集、実行を処理するためのプログラム上のクラスの名前である。このタスククラスからＷＥＢアプリケーション302は予約タスクの編集画面操作のための処理、スケジューラ303は予約タスクから後述のタスク情報とサブタスク情報を作成する処理、エージェント103はタスクの実行処理を実装するクラスを動的に作成する。
【００４０】
タスク名は、ユーザがテキストボックス401に入力した、ユーザが各予約タスクを識別するためにユーザが付けた名前である。タスク詳細は、タスクの詳細を、各タスククラスが解釈できる形で保存したデータである。例えば、収集する設定値を指定できる場合は、ここに取得対象の設定値の名前のリストを保存する。表３の例では、["network","security"]と記載されており、ネットワーク設定とセキュリティ設定の取得を指定している。
【００４１】
スケジュール設定は、タスクの実行スケジュールである。例では、毎週の繰り返しが指定されている（曜日と時刻指定の詳細は省略）。
【００４２】
開始日時は、スケジュール設定に従った場合に、次に予約タスクを実行すべき日時である。例えば、図４で例示するスケジュール設定（毎週金曜日の午前８時）で、今日が水曜日の場合、開始日時は次の金曜日の午前８時となる。なお、タスクの対象となるネットワーク機器は、データベース上の別のテーブル（例えば、予約タスクＩＤとネットワーク機器を一意に識別するＩＤとを列に持つテーブル（不図示））で管理されるものとする。即ち表３に示す予約タスク情報は、予約タスクＩＤによって、タスクの対象となるネットワーク機器のデバイスＩＤが対応付けられている。このように、表３に示す予約タスク情報は、対象となるネットワーク機器のデバイスＩＤが指定された予約タスク情報となる。
【００４３】
図５（ａ）は、スケジューラ303がＤＢ112内の予約タスクの内の、実行予定日時が現在時刻より以前の予約タスクを開始する処理（タイマー処理）の一例を示すフローチャートである。図５（ａ）及び後述する図５（ｂ）の処理は、管理装置101のＣＰＵ201が外部記憶装置207に格納されたプログラムをＲＡＭ202にロードして実行することにより実現

される管理アプリケーション101aにより実行される。
【００４４】
スケジューラ303は、起動時にオペレーティングシステム（ＯＳ）やアプリケーションの実行環境（例えばJREや.NET Framework）が提供するタイマーに対し、図５（ａ）で示す管理アプリケーション101aのタイマー処理を定期的に呼び出すように指示する。これにより、タイマー処理が開始される。
【００４５】
タイマー処理では、管理アプリケーション101aは、先ずS501において、ＤＢ112から実行予定時日時が現在時刻より以前の予約タスクの一覧を取得する。
次にS502において、管理アプリケーション101aは、上記S501で取得した予約タスクの一覧から未処理の予約タスクを１つ取り出し、S503に処理を進める。
【００４６】
S503にて、管理アプリケーション101aは、上記S502で一覧からの予約タスクの取り出しに成功したかの確認を行う。管理アプリケーション101aは、一覧からの予約タスクの取り出しが成功したと判定した場合（S503でYesの場合）、S504に処理を進める。
【００４７】
S504において、管理アプリケーション101aは、上記S502で取り出した予約タスクのタスククラスに従い、該予約タスクの開始処理を実装したクラスのクラスインスタンスを作成し、該クラスインスタンスの予約タスクの開始処理を呼び出す。予約タスクの開始処理については図５（ｂ）を用いて後述する。
【００４８】
次にS505において、管理アプリケーション101aは、上記S502で取り出した予約タスクのスケジュール設定に従い次の実行予定日時を計算し、ＤＢ112内の該予約タスクのデータを更新する。ここで、一度限りの実行や、定期実行の終了日が設定されている等の理由により次の実行予定日時が存在しない場合、タイマー処理は遠い未来（例えば、9999年1月1日）を実行予定日時として設定する。予約タスクのデータの更新後は、管理アプリケーション101aは、S502に処理を遷移し、次の予約タスクに対する処理に移行する。
【００４９】
また、上記S503において、管理アプリケーション101aは、上記S502で一覧からの予約タスクの取り出しに失敗したと判定した場合（S503でNoの場合）、即ちS501で取得された全ての予約タスクについて処理が終了した場合、タイマー処理を終了する。
【００５０】
図５（ｂ）は、図５（ａ）のタイマー処理のS504で呼び出される予約タスクの開始処理（以下「予約開始処理」）のフローチャートである。予約開始用のクラスは、以下に示すTaskRunner抽象クラスを継承する。予約開始用のクラスは、Startという関数を実装している。Start関数は、予約タスク情報を引数に取る。
【００５１】
```
public abstract class TaskRunner {
public void Start( ReservedTask task );
}
```
【００５２】
予約開始処理では、管理アプリケーション101aは、S510において、上記引数として渡された予約タスク情報（task）から、新規にタスク情報を作成し、ＤＢ112に格納する。
表４に、予約開始処理がS510にて作成するタスク情報の一例を示す。
【００５３】

【表４】

タスクID	b7f35ba6-e908-47d5-8e22-d140474b31e6
予約タスクID	5e16e760-f275-4a82-bcd4-a9ee273af6f9
タスククラス	GetSettings
タスク名	設定値の定期取得
タスク詳細	["network", "security"]
ステータス	RUNNING
開始日時	2017-04-2916:23:18Z
終了日時	

【００５４】
表４において、タスクＩＤは、このタスク情報を一意に識別するため予約開始処理が付与した識別子であり、ここではＵＵＩＤを使用している。予約タスクＩＤ、タスククラス、タスク名、タスク詳細は、タイマー処理（図５（ａ））のS504で予約開始処理に渡した予約タスク情報の予約タスクＩＤ、タスククラス、タスク名、タスク詳細である。これらは、タスク情報の作成時点では予約タスク情報と同じものであるが、予約タスクはタスク情報作成後にユーザにより変更や削除がなされることもあるため、タスク情報内に同データを持つことにしている。
【００５５】
ステータスは、このタスク情報により表現されるタスクの状態を示す。作成時点では「実行中」である。タスクの状態には、「実行中」、「正常（全ての対象のネットワーク機器に対する処理が成功した）」及び「失敗（処理に失敗したネットワーク機器が存在する）」がある。開始日時は、このタスク情報が作成された日時を示す。終了日時は、このタスク情報により表現されるタスクの実行が終了した日時を示す。タスク情報の作成時点では、終了日時は空である。
【００５６】
なお、表３の説明に記載したように、タスクの対象となるネットワーク機器は、データベース上の別のテーブル（例えば、予約タスクＩＤとネットワーク機器を一意に識別するＩＤ（デバイスＩＤ）とを列に持つテーブル（不図示））で管理される。即ち表４に示すタスク情報は、予約タスクＩＤによって、タスクの対象となるネットワーク機器のデバイスＩＤが対応付けられている。このように、表４に示すタスク情報は、対象となるネットワーク機器のデバイスＩＤが指定されたタスク情報である。
【００５７】
次にS511において、管理アプリケーション101aは、予約タスクの対象となるネットワーク機器の一覧を取得する。例えば、管理アプリケーション101aは、予約タスクＩＤを用いて、ＤＢ112内の不図示のテーブル（予約タスクＩＤとネットワーク機器を一意に識別するＩＤとを列に持つテーブル）から、予約タスクの対象となるネットワーク機器の一覧を取得する。
【００５８】
次にS512において、管理アプリケーション101aは、上記S511で取得したネットワーク機器情報の一覧から未処理のネットワーク機器情報を１つ取り出す。
次にS513において、管理アプリケーション101aは、上記S511で取得した一覧からネットワーク機器情報の取り出しに成功したか否かを確認する。管理アプリケーション101aは、一覧からのネットワーク機器情報の取り出しが成功したと判定した場合（S513でYesの場合）、S514に処理を進める。

【００５９】
S514において、管理アプリケーション101aは、上記S512で取り出したネットワーク機器情報に対応するサブタスク情報を作成し、これをＤＢ112に格納した後、S512に処理を遷移する。ここでサブタスクとは、タスクを、対象の各ネットワーク機器に対する処理に分解した処理の単位を示す。
表５に、予約開始処理がS514にて作成したサブタスク情報の一例を示す。
【００６０】
【表５】

サブタスクID	729bae26-beb5-4e2a-998f-8baf16af1644
タスクID	b7f35ba6-e908-47d5-8e22-d140474b31e6
デバイスID	41124cd1-74f8-4191-8c39-f8027c9c05a2
エージェントID	e887bddb-f3a9-404a-a7b6-e27576d0975b
ｻﾌﾞﾀｽｸ詳細	
ステータス	READY
実行結果	
開始日時	
終了日時	

【００６１】
サブタスクＩＤは、このサブタスク情報を一意に識別するために予約開始処理が付与した識別子であり、ここではＵＵＩＤを使用している。タスクＩＤは、図５（ａ）のS501で作成したタスク情報を一意に識別する識別子、即ち表４のタスクＩＤである。デバイスＩＤは、このサブタスク情報の対象となるネットワーク機器を一意に識別する識別子、即ち表２のデバイスＩＤである。エージェントＩＤは、対象のネットワーク機器に紐づいたエージェントを一意に識別する識別子、即ち表１及び表２のエージェントＩＤである。
【００６２】
サブタスク詳細は、このタスクを処理する各クラスが解釈可能な、このサブタスク情報に固有の詳細である。この例では、サブタスク情報固有の詳細は存在しないので空となっている。ステータスは、このサブタスク情報で表現されるサブタスクの状態を示す。サブタスク情報作成時点では、サブタスクの実行は開始されていないため「準備中」である。ステータスには「準備中」、「実行中」、「成功」、「失敗」、「例外（実行中に予期せぬエラーが発生した）」及び「キャンセル」がある。
【００６３】
実行結果は、このタスクを処理する各クラスが解釈可能な、このサブタスクの実行結果である。作成時点では、実行が開始されていないため、実行結果は空である。開始日時、終了日時は、それぞれサブタスクのエージェントによる実行の開始の日時、終了の日時である。即ち、表５に示すサブタスク情報は、エージェントＩＤ及び対象デバイスの一部のデバイスＩＤが指定されたサブタスク情報である。
【００６４】
また、上記S513において、管理アプリケーション101aは、上記S511で取得した一覧からネットワーク機器情報の取り出しに失敗したと判定した場合（S513でNoの場合）、即ち上記S511で取得したネットワーク機器の全てに対する処理が終了した場合、S515に処理を進める。
S515において、管理アプリケーション101aは、タスクの開始を各エージェントに通知し

、予約開始処理を終了する。エージェント103aへのタスクの開始指示については後述する。

【００６５】

図６は、エージェント103aの構成を例示する図である。即ち、図６に示すエージェント103aの構成は、監視装置103のＣＰＵ201が外部記憶装置207に格納されたエージェント103aに対応するプログラムをＲＡＭ202にロードして実行することにより実現される。

【００６６】

図６において、受信部601は、管理アプリケーション101aからの指示を受信する。受信部601は、例えば、ＨＴＴＰ／ＨＴＴＰＳサーバを内蔵し、管理アプリケーション101aからのＨＴＴＰ要求を受信して、要求のＵＲＬやメソッドにより所定の処理を実行する。予約開始処理（図５（ｂ））のS515の「エージェント103aへのタスクの開始通知」は、この受信部601が受信する（詳細は後述する）。

【００６７】

デバイス管理部602は、本実施例の分散ネットワーク機器管理システムにより管理されているネットワーク機器のうち、該エージェント103aに紐づいているネットワーク機器の情報を、管理アプリケーション101aから取得し、これを管理する。デバイス管理部602は、例えばデバイスＩＤを用いてネットワーク機器の情報を検索する機能を有する。これにより、デバイスＩＤからネットワーク機器のＩＰアドレスを取得する等が可能になる。デバイス管理部602は更に、後述する通信部605がネットワーク機器と通信する際の認証情報を管理する機能を有する。例えば、ＳＮＭＰバージョン３を用いてネットワーク機器に接続する際のユーザ名、コンテキスト名、認証プロトコルとキー、暗号化プロトコルとキー等の情報を管理する。

【００６８】

送信部603は、指定されたデータを、管理アプリケーション101aの指定されたアドレス（ＵＲＬ）に送信する機能を有する。また送信部603は、管理アプリケーション101aの指定されたアドレス（ＵＲＬ）に送信された要求に対する応答を、送信依頼元に返す機能を有する。

【００６９】

タスク実行部604は、管理アプリケーション101aから実行を通知されたタスクを実行する。通信部605は、タスク実行部604からの依頼により、ネットワーク機器107との通信を行う。通信部605は、例えばネットワーク機器107とＳＮＭＰやＷＥＢサービス等を用いて通信を行う。

【００７０】

以下、図７（ａ）、図７（ｂ）を用いて、エージェント103aのタスク実行部604によるタスクの実行処理の流れについて説明する。

図７（ａ）は、予約開始処理（図５（ｂ））のS515のタスクの開始通知により実行されるタスク起動処理の流れを示すフローチャートである。図７（ａ）及び後述する図７（ｂ）、図８、図９に示す処理は、監視装置103のＣＰＵ201が外部記憶装置207に格納されたプログラムをＲＡＭ202にロードして実行することにより実現されるエージェント103aにより実行される。

【００７１】

予約開始処理（図５（ｂ））のS515では、管理アプリケーション101aはエージェント103aに対して、例えば以下のようにタスクの開始を通知する。予約開始処理は、ＵＲＬパスが「/API/Task/Start」、ＨＴＴＰメソッドがＰＯＳＴ，本体が以下のＪＳＯＮデータであるＨＴＴＰ要求を、エージェント103aに対して送信することでタスクの開始通知を行う。

【００７２】

```
{ "taskid": "b7f35ba6-e908-47d5-8e22-d140474b31e6",
"taskclass": "GetSettings" }
```

【００７３】

エージェント103aの受信部601は、上記ＪＳＯＮデータを受信すると、該受信したＪＳＯＮデータをオブジェクトに変換して、これを引数にして、図７（ａ）のタスク起動処理を呼び出す。

【００７４】

タスク起動処理において、エージェント103aのタスク実行部604は、S701において、上記引数として渡されたオブジェクト内のタスククラス属性から、タスク実行用のクラスを決定し、このクラスのインスタンスを生成する。タスク実行用のクラスは、以下に示すTaskBase抽象クラスを継承する。タスク実行用のクラスはStartとOnSubtaskEndという２つの関数を実装している。タスク起動処理では、タスク実行用のクラスのインスタンスを作成した際に、上記引数として渡されたオブジェクト内のタスクＩＤを、作成したインスタンスのTaskIdに設定する。

【００７５】

```
public abstract class TaskBase {
public UUID TaskId;
public void Start( );
public abstract void OnSubtaskEnd( SubtaskInfo result );
}
```

【００７６】

次にS702において、タスク実行部604は、上記S701で作成したタスク実行用クラスのインスタンスのStart関数を呼び出す。

【００７７】

図７（ｂ）は、タスク起動処理のS702で呼び出される、タスク実行用クラスのStart関数の処理（タスク開始処理）の流れを示すフローチャートである。

【００７８】

タスク開始処理では、タスク実行部604は、まずS710において、インスタンスに設定されたタスクＩＤ（TaskId）で指定されるタスク情報（表４を参照）を、送信部603を用いて管理アプリケーション101aから取得し、タスク情報内のタスク詳細を解釈する。

【００７９】

次にS711において、タスク実行部604は、送信部603を用いて上記タスクＩＤとエージェント103aに紐づいたサブタスク情報（表５を参照）の一覧を、管理アプリケーション101aから取得する。

【００８０】

次にS712にて、タスク実行部604は、上記S711で取得したサブタスク情報一覧から、未処理のサブタスク情報を１つ取り出し、S713に処理を進める。

S713において、タスク実行部604は、上記S712でサブタスク情報の取り出しに成功したか否かを判定する。タスク実行部604は、上記S712でサブタスク情報の取り出しに成功したと判定した場合（S713でYesの場合）、S714に処理を進める。

【００８１】

S714において、タスク実行部604は、上記S712で取得したサブタスク情報と上記S710で取得したタスク詳細とから、サブタスク実行用クラスのインスタンスを作成・設定し、これをサブタスクキュー（図示しない）に積む。なお、サブタスク実行用クラスについては後述する。その後、タスク実行部604は、S712に遷移し、次のサブタスク情報に処理を移行する。

【００８２】

また、上記S713において、タスク実行部604は、上記S712でサブタスク情報の取り出しに失敗したと判定した場合（S713でNoの場合）、即ち全てのサブタスク情報に対する処理が完了した場合、タスク開始処理を終了する。

【００８３】

なお、上述のサブタスク実行用クラスは、下記に示す抽象クラスを実装する。タスク開始処理（図７（ｂ））のS714では、作成したサブタスク実行用のクラスのインスタンスに、

タスク実行用のクラスインスタンス（図７（ａ）のS701で作成されたインスタンス）と、サブタスク情報に含まれるサブタスクＩＤと、デバイスＩＤとを設定する。さらに、後述するサブタスク実行処理では、必要に応じて各タスクの処理に固有の情報を、サブタスク実行用クラスのインスタンスに設定する。

【００８４】

```
public abstract class SubtaskBase {
public TaskBase Task;
public UUID SubtaskId;
public UUID DeviceId;
public abstract SubtaskInfo Start( );
}
```

【００８５】

タスク実行部604は、サブタスクキュー処理として、サブタスクキューから、タスク開始処理（図７（ｂ））のS714でサブタスクキューに積まれたサブタスク実行用クラスのインスタンスを順次取り出す。さらに、タスク実行部604は、該取り出したサブタスクのインスタンスを引数にして、図８に示すサブタスク実行処理を呼び出し、サブタスクを実行する。

【００８６】

図８は、サブタスク実行処理の処理の流れを示すフローチャートである。

サブタスク実行処理では、タスク実行部604は、先ずS801において、引数として渡されたサブタスク実行用クラスのインスタンスのStart関数を呼び出し、サブタスク実行結果を受け取る。

【００８７】

サブタスク実行用のクラスのStart関数では、先ず、デバイスＩＤを用いて、デバイス管理部602から対象のネットワーク機器に関する情報（例えばＩＰアドレスや各プロトコルに対する認証情報等）を取得する。続いてサブタスク実行用のクラスのStart関数では、取得したネットワーク機器に関する情報を用いて、ネットワーク機器に対して要求された処理を実行し、サブタスク実行結果を返す。サブタスクの実行結果は、表５に示すサブタスク情報と同等である。

【００８８】

次にS802において、タスク実行部604は、上記S801で受け取ったサブタスク実行結果の中の、タスクＩＤ、サブタスクＩＤ、デバイスＩＤ、開始日時、終了日時を更新する。開始日時は、上記S801でサブタスク実行クラスのStart関数を呼び出す前の日時となる。また、終了日時は、現在日時となる。サブタスク詳細、ステータス、実行結果は、サブタスク実行クラスのStart関数により設定される。例えば、ネットワーク機器の設定情報の取得タスクの場合、ネットワーク機器から取得した設定情報を表現するＪＳＯＮ文字列が実行結果となる。同様に、状態や消耗品情報の取得タスクの場合は、ネットワーク機器から取得した状態や消耗品情報を表現するＪＳＯＮ文字列が実行結果となる。

【００８９】

最後にS803にて、タスク実行部604は、サブタスク実行クラスのインスタンのTask属性（即ちタスク実行用のクラスのインスタンス）のOnSubtaskEnd関数を呼び出すことで、サブタスクの実行の終了をタスク実行用クラスに通知する。タスク実行クラスのOnSubtaskEnd関数の典型的な処理は、渡されたサブタスクの実行結果を、管理アプリケーション101a内のＷＥＢアプリケーション302のサブタスク実行結果の受信用のＵＲＬに対して送信するように、送信部603に依頼することである。ＷＥＢアプリケーション302のサブタスク実行結果受信用のＵＲＬの例を以下に示す。ここでは、管理アプリケーション101aの動作するホストのアドレスを172.20.1.11と仮定する。

【００９０】

```
https://172.20.1.11/api/v1/tasks/subtask/GetSettings
```

【００９１】

即ち、タスク実行クラスのOnSubtaskEnd関数は、このＵＲＬに対して、ＨＴＴＰメソッドがＰＯＳＴ、本文がサブタスク実行結果をＪＳＯＮ文字列に変換した文字列である、ＨＴＴＰ要求を送信するよう、送信部603に依頼する。

【００９２】

図９は、エージェント103aの送信部603から送信された前記サブタスク実行結果をＷＥＢアプリケーション302が受信した際の処理（サブタスク実行結果受信処理）の流れを示すフローチャートである。

【００９３】

ＷＥＢアプリケーション302は、ＵＲＬパス「/api/v1/tasks/subtask/クラス名」に対するＰＯＳＴメソッドのＨＴＴＰ要求を受信した際に、図９に示すサブタスク実行結果受信処理を、「クラス名」と要求の本文に含まれるサブタスク実行結果を引数にして呼び出す。例えば、前記ＵＲＬの例では、「クラス名」は「GetSettings」である。

【００９４】

サブタスク実行結果受信処理では、管理アプリケーション101aは、先ずS901において、引数で与えられたクラス名を用いて、このサブタスク受信結果を処理可能なサブタスク実行結果処理用クラスのインスタンスを作成する。サブタスクの実行結果を処理するクラスは、以下に示すインターフェイスを実装したクラスである。

【００９５】

```
public interface SubtaskResultHooker {
SubtaskInfo Hook(SubtaskInfo result);
}
```

【００９６】

次にS902において、管理アプリケーション101aは、引数とした渡されたサブタスク実行結果を引数にして、作成したサブタスク実行結果処理用クラスのインスタンスのHook関数を呼び出し、結果を受け取る。

【００９７】

サブタスク実行結果処理用クラスのHook関数では、引数として渡されたサブタスク実行結果を基に各タスクに固有の処理を行い、必要に応じてサブタスク実行結果の内容を変更し、これを返して関数を終了する。ここで各タスクに固有の後処理の例を以下に示す。例えば、ネットワーク機器の状態の取得タスクの場合、ネットワーク機器の現在の状態を格納するＤＢ112内のテーブルの内容を更新し、前回の状態から新たにエラー状態が検出された場合、その旨をユーザが指定したメールアドレスに対してメール送信する。例えば、ネットワーク機器の利用状況（例えばネットワーク機器が画像形成装置である場合、各種カウンタの値）の取得の場合、利用状況の統計を格納するＤＢ112内のテーブルの内容を更新する。

さらに、Hook関数では、関数に渡されたサブタスク実行結果が、サブタスク実行結果としてＤＢ112内のサブタスク情報テーブルに保存する必要のない情報であれば、サブタスク実行結果内の実行結果属性を空にして呼び出し元に返す。

【００９８】

次にS903において、管理アプリケーション101aは、Hook関数の戻り値を判定する。管理アプリケーション101aは、Hook関数の戻り値が空と判定した場合（S903でYesの場合）、サブタスク実行結果受信処理を終了する。

【００９９】

一方、上記S903において、管理アプリケーション101aは、Hook関数の戻り値が空でないと判定した場合（S903でNoの場合）、S904に処理を進める。

S904において、管理アプリケーション101aは、Hook関数から返されたサブタスク実行結果を用いて、表５に示すＤＢ112内のサブタスク情報を更新する。

【０１００】

次にS905において、管理アプリケーション101aは、受信したサブタスク実行結果のタスクＩＤに紐づいた全てのサブタスクの実行が終了したか否かを判定する。例えば、管理ア

プリケーション101aは、ＤＢ112内の、受信したサブタスクに紐づいたタスクＩＤと同じタスクＩＤを持つサブタスク情報のステータスが終了状態（成功、失敗、例外、キャンセル）でないサブタスク情報の数を取得する。そして、該取得した数が０の場合、受信したサブタスク実行結果のタスクＩＤに紐づいた全てのサブタスクの実行が終了したと判定する。一方、該取得した数が０でない場合、受信したサブタスク実行結果のタスクＩＤに紐づいたサブタスクのうち、まだ実行が終了していないサブタスクが存在すると判定する。

【０１０１】

上記S905において、管理アプリケーション101aは、受信したサブタスク実行結果のタスクＩＤに紐づいたサブタスクのうち、まだ実行が終了していないサブタスクが存在すると判定した場合（S905でNoの場合）、サブタスク実行結果受信処理を終了する。

【０１０２】

一方、上記S905において、管理アプリケーション101aは、受信したサブタスク実行結果のタスクＩＤに紐づいた全てのサブタスクの実行が終了したと判定した場合（S905でYesの場合）、S906に処理を進める。

【０１０３】

S906において、管理アプリケーション101aは、表４に示すＤＢ112内のタスク情報のステータスと終了日時を更新する。ステータスについては、タスクＩＤを持つ全てのサブタスク情報のステータスが「成功」の場合は「成功」、そうでない場合は「失敗」となる。S906の処理の後、管理アプリケーション101aは、サブタスク実行結果受信処理を終了する。

【０１０４】

図１１は、管理アプリケーション101aによるタスクの開始処理から、エージェント103aによるタスクの実行と、管理アプリケーション101aへの結果の送信までの流れを示すシーケンス図である。このシーケンス図において、管理アプリケーション101aの処理は、管理装置101のＣＰＵ201が外部記憶装置207に格納されたプログラムをＲＡＭ202にロードして実行することにより実現される管理アプリケーション101aにより実行される。また、エージェント103aの処理は、監視装置103のＣＰＵ201が外部記憶装置207に格納されたプログラムをＲＡＭ202にロードして実行することにより実現されるエージェント103aにより実行される。

【０１０５】

まず管理アプリケーション101aは、図５（ａ）及び図５（ｂ）の予約開始処理1101を実行する。予約開始処理1101は、対象デバイスのデバイスＩＤが対応付けられたタスク情報を生成し、エージェントＩＤ及び対象デバイスの一部のデバイスＩＤが指定されたサブタスク情報を複数生成し、さらにタスク開始要求1102をエージェント103aに送信する。なお、タスク開始要求1102は、エージェント104aを含む複数のエージェントに送信されるが、説明を容易にするために図１１に図示していない。

【０１０６】

エージェント103aは、タスク開始要求1102を受信部601が受信し、図７（ａ）のタスク実行処理1103を実行する。タスク実行処理1103は、タスク開始要求1102に含まれるタスク実行用クラスのインスタンスを作成し、図７（ｂ）のタスク開始処理1104を呼び出す。

タスク開始処理1104は、まずタスク開始要求1102に含まれるタスク情報ＩＤを持つタスク情報の取得要求1105を管理アプリケーション101aに送信する。

管理アプリケーション101aは、タスク情報取得要求1105に含まれるタスク情報ＩＤを持つタスク情報をＤＢ112から取得し、これを返信する（1106）。

【０１０７】

タスク開始処理1104は、次に、タスク開始要求1102に含まれるタスク情報ＩＤと自身のエージェントＩＤとを持つサブタスク情報の取得要求1107を管理アプリケーション101aに送信する。

管理アプリケーション101aはサブタスク情報取得要求1107に含まれるタスク情報ＩＤと

エージェントＩＤとを持つサブタスク情報の一覧をＤＢ112から取得しこれを返信する（1108）。

【０１０８】

タスク開始処理1104は、取得したタスク情報とサブタスク情報とから、サブタスクを実行するためのクラスを生成・設定しサブタスク実行処理1109を呼び出す。サブタスク実行処理1109は、必要に応じて指定された管理対象のネットワーク機器と通信し、サブタスクの実行結果1110を、管理アプリケーション101aに送信する。より正確には、サブタスク実行処理1109が返すサブタスク実行結果1110を、タスク開始処理1104が、管理アプリケーション101aに送信する。

【０１０９】

管理アプリケーション101aは、サブタスク実行結果1110を受信して、図９のサブタスク実行結果受信処理1111を呼び出す。サブタスク実行結果受信処理1111は、サブタスク実行結果1110に含まれるクラス名からサブタスク実行結果処理用クラスを生成し、サブタスク実行結果処理（即ちHook関数）1112を呼び出す。サブタスク実行結果処理1112は、最後に、サブタスク実行結果処理1112が返すサブタスク実行結果の内容で、ＤＢ112内のサブタスク情報を更新する。なお、上記管理アプリケーション101aとエージェント103aとの通信に使用される各ＵＲＬ及び各要求の受信処理は、タスクの種類に依らず共通である。

【０１１０】

以上のように、実施例１では、管理アプリケーション101aは、処理をネットワーク機器毎のサブタスクに分割し、タスクの開始を通知する。エージェント103aは、各サブタスクを実行し、実行結果を管理アプリケーション101aに送信する。管理アプリケーション101aは、受信したサブタスクの実行結果に基づき前記サブタスク情報を更新し、全てのサブタスク情報を更新されたら、タスク情報を更新する。このような構成により、ネットワーク機器を分散管理するシステムにおいて、管理アプリケーション101aからエージェント103aに、ネットワーク機器に対する処理（例えば設定値の取得処理）を実行させることができる。

【０１１１】

上記説明では、デバイスから情報（例えば設定値や稼動情報）を取得する処理について説明したが、例えば、デバイスに設定値を配信する処理や、デバイスにアプリケーションソフトウェアを配信する処理などでも同様である。

【実施例２】

【０１１２】

実施例２では、タスクがネットワーク機器を探索する処理（探索処理）のタスクである場合について説明する。ここで探索処理は、指定したコミュニティ名を持つＳＮＭＰのバージョン１の要求を、指定したＩＰアドレスに送信し、要求に対する応答を確認することで、ネットワーク機器の接続の有無を判断する。また、ＳＮＭＰのバージョン1のコミュニティ名はタスク全体で共通とし、要求を送信するＩＰアドレスの範囲はエージェント毎に指定するものとする。

【０１１３】

表６に、タスクがネットワーク機器の探索である場合のタスク情報の一例を示す。

表７に、タスクがネットワーク機器の探索である場合のサブタスク情報の一例を示す。

【０１１４】

【表６】

タスクID	23b1d708-9bbc-4279-81fa-83804002f3c1
予約タスクID	14aa2388-b364-442f-9767-b38ce821e612
タスククラス	Discovery
タスク名	ネットワーク機器の探索
タスク詳細	{"communities": ["public", "secret"]}
ステータス	RUNNING
開始日時	2017-04-29 16:23:18Z
終了日時	

【０１１５】
【表７】

サブタスクID	2e7eacc7-00c8-452e-8fec-cb7762e6e438
タスクID	23b1d708-9bbc-4279-81fa-83804002f3c1
デバイスID	
エージェントID	e887bddb-f3a9-404a-a7b6-e27576d0975b
ｻﾌﾞﾀｽｸ詳細	{"addresses":"172.16.0.0-172.16.255.255"}
ステータス	READY
実行結果	
開始日時	
終了日時	

【０１１６】
以下、表６、表７において、表４、表５と異なる箇所のみ説明する。
表６において、タスククラスは、探索処理を実装したクラス名である「Discovery」になっている。タスク名は、「ネットワーク機器の探索」となっている。また、タスク詳細は、ＳＮＭＰのバージョン1を利用して探索を行う際のＳＮＭＰのバージョン１のコミュニティ名の配列となっている。探索処理は、ここに列挙されたコミュニティ名を用いてネットワーク端末とのＳＮＭＰのバージョン１の通信を試行する。探索処理は、ネットワークに接続されたネットワーク機器を見つける処理であるため、サブタスクの対象はエージェントとなり、タスクの対象となるネットワーク機器の情報は存在しない。なお、タスクの対象となるエージェントは、データベース上の別のテーブル（例えば、予約タスクＩＤとエージェントＩＤとを列に持つテーブル（不図示））で管理されるものとする。即ち表６に示す実施例２の予約タスク情報は、予約タスクＩＤによってタスクの対象となるエージェントＩＤが対応付けられている。このように、実施例２の予約タスク情報は、対象となるエージェントＩＤが指定された予約タスク情報となる。また、表７において、サブタスク情報のデバイスＩＤは空になっている。また、探索処理の場合、サブタスクは、処理対象のエージェントごとに生成される。サブタスク詳細は、エージェント上で実行される探索処理が、探索用のＳＮＭＰのバージョン1のパケットを送信する宛先となっている。探

索処理は、ここに記載されたネットワークアドレスのそれぞれに対して、探索用のＳＮＭＰのバージョン1のパケットを送信することで、ネットワーク機器の探索を行う。表７の例では、172.16.0.0～172.16.255.255の合計65536個のＩＰアドレスのそれぞれに対してＳＮＭＰパケットを送信することを意味する。

【０１１７】

タスクがネットワーク機器の探索の場合、図７（ａ）のS701でタスク起動処理により呼び出される、エージェント103a上で動作する探索タスク実行クラスのStart関数は、探索処理を実行し、途中経過を一定時間毎に管理アプリケーション101aに送信する。エージェント103a上で動作する探索処理が、探索処理の途中経過をサブタスク実行結果として管理アプリケーション101aに送信する際の実行結果の一例を以下に示す。

【０１１８】

{"scanned": 18272,
"discovered": 18,
"devices": [
{"ip": "172.16.95.117", "name": "MFP-6500",
"mac": "F4:81:28:C8:47:91" },
{"ip": "172.16.110.181", "name": "SFP-2000",
"mac": "F4:81:19:E2:B1:C9" },
{"ip": "172.16.113.134", "name": "SFP-3500",
"mac": "F4:81:23:AC:BF:45" },
{"ip": "172.16.138.91", "name": "MFP-5500",
"mac": "F4:81:92:BE:F1:1A" }
}]
}

【０１１９】

ここで、scannedはその時点までに探索を実行したアドレスの数、discoveredは見つかったネットワーク機器の数、devicesは前回の途中経過の送信から新たに見つかったネットワーク機器の情報のリストである。なお、この時点では、探索対象の全てのＩＰアドレスに対する処理が完了していないので、送信するサブタスク実行結果のステータスは「RUNNING」となっている（不図示）。

【０１２０】

タスクがネットワーク機器の探索の場合、図９のS902でサブタスク実行結果受信処理により呼び出される実行結果処理クラスの実行結果処理関数は、以下の動作を行う。

【０１２１】

先ず、実行結果処理関数は、サブタスクの実行結果から、探索されたネットワーク機器情報の一覧（前記「devices」情報）を取り出す。次に、実行結果処理関数は、取り出したネットワーク機器情報のそれぞれに対し、ＤＢ112内のネットワーク機器情報のテーブルに同じＭＡＣアドレスを持つエントリがあるか否かを判定する。該判定の結果、存在する場合、実行結果処理関数は、ＤＢ112内のテーブル内の情報を受信した情報で更新する。存在しない場合、実行結果処理関数は、受信したネットワーク機器情報を、新規にＤＢ112のネットワーク機器情報テーブルに追加する。そして、受信した全てのネットワーク機器情報に対する処理が完了したら、実行結果処理関数は、引数として渡されたサブタスク実行結果データ内の実行結果から「devices」を削除したものを新たな実行結果としたサブタスク実行結果を、関数の呼び出し元に返す。

【０１２２】

実行結果処理関数が終了すると、サブタスク実行結果受信処理は、実行結果処理関数から返されたサブタスク実行結果によりサブタスク情報を更新する（S903、S904）。この際、サブタスク実行結果受信処理は、そのサブタスクにおける探索対象の全てのＩＰアドレスに対する処理が完了していたら、サブタスク情報のステータスを終了状態に更新する。さらに、サブタスク実行結果受信処理は、全サブタスクが終了していたら、対応するタス

ク情報のステータスと終了日時を更新する（S905、S906）。
【０１２３】
以上、実施例２によれば、ネットワーク機器を分散管理するシステムにおいて、管理アプリケーション101aからエージェント103aに、ネットワーク機器に対する処理（例えばネットワーク機器の探索処理）を実行させることができる。
【実施例３】
【０１２４】
実施例３では、タスクが定期的な管理対象のネットワーク機器のステータス監視の場合について説明する。ネットワーク機器の定期的なステータス監視では、管理アプリケーション101aからの通知に依らずに、エージェント103aが、自身に紐づいたネットワーク機器からステータス情報を取得し、これを管理アプリケーション101aに送信する。従って、ＤＢ112のタスク情報やサブタスク情報を格納するテーブルには、ステータス監視用のタスク情報とサブタスク情報は存在しない。
【０１２５】
TaskBase抽象クラスを継承したタスク実行クラスには、エージェント起動時に実行する旨を示す属性（「AutoStart」とする）を持つものがある。エージェント103aは、起動時に、タスク実行クラス、即ちTaskBase抽象クラスを継承したクラスの中から前記AutoStart属性を持つクラスの一覧を動的に取得し、それらクラスのインスタンスを生成しStart関数を呼び出す。ステータス監視用のTaskBase抽象クラスを継承したタスク実行クラスは、AutoStart属性を持つため、エージェント103aは、起動時にステータス監視用クラスのインスタンスを作成し、Start関数を呼び出す。
【０１２６】
ステータス管理タスククラスのStart関数は、ＯＳやアプリケーション実行環境（例えばJREや.NET Framework）の提供するタイマーに一定時間毎に、監視開始処理を呼び出すように設定する。
【０１２７】
監視開始処理は、管理アプリケーション101aから、エージェント103aに紐づいたネットワーク機器の一覧を取得する。そして、監視開始処理は、該取得したネットワーク機器毎に、ステータス監視用のサブタスク実行クラスのインスタンスを作成し、作成したインスタンスをサブタスクキューに追加する。
【０１２８】
SubtaskBase抽象クラスを実装するステータス監視用のサブタスク実行クラスのStart関数は、サブタスク情報で指定されたネットワーク機器からスタータス情報を取得し、サブタスク実行結果を返す。
表８に、ステータス監視用のサブタスク実行クラスのStart関数が返すサブタスク実行結果の一例を示す。
【０１２９】

【表８】

サブタスクID	00000000-0000-0000-0000-000000000000000000000
タスクID	00000000-0000-0000-0000-000000000000000000000
デバイスID	41124cd1-74f8-4191-8c39-f8027c9c05a2
エージェントID	e887bddb-f3a9-404a-a7b6-e27576d0975b
ｻﾌﾞﾀｽｸ詳細	
ステータス	SUCCESS
実行結果	{"alerts": [...], "supplies": [...], "inputs": [...]}
開始日時	2017-04-29 17:19:38Z
終了日時	2017-04-29 17:21:02Z

【０１３０】
以下、表８において、表５と異なる箇所のみ説明する。
表８において、実行結果は、ネットワーク機器のエラー情報と消耗品情報と給紙情報のリストより構成される（詳細は省略する）。サブタスクＩＤとタスクＩＤは、空のＵＵＩＤである。
【０１３１】
このサブタスク実行結果がエージェント103aから管理アプリケーション101aのＷＥＢアプリケーション302に送信され、図９のサブタスク実行結果受信処理がこれを受信する。図９のS902でサブタスク実行結果受信処理は、SubtaskResultHookerインターフェイスを実装したステータス監視用のサブタスク受信結果処理クラスのHook関数を呼び出す。Hook関数は、受信したサブタスク実行結果の実行結果からステータス情報を生成し、ＤＢ112内のネットワーク機器のステータス保存用のテーブルにステータス情報を保存し、空の実行結果を返す。空の結果を受け取った図９のサブタスク実行結果受信処理は、S903の判定処理を経て受信処理を終了する。
【０１３２】
以上、実施例３によれば、ネットワーク機器を分散管理するシステムにおいて、エージェント103aに、ネットワーク機器に対する処理（例えばネットワーク機器のステータス監視処理）を実行させることができる。
【０１３３】
以上、実施例１～３によれば、ネットワーク機器を分散管理するシステムにおいて、処理に依存せず、エージェントに効率的にネットワーク機器に対する処理を実行させることができる。
【実施例４】
【０１３４】
実施例４は、上述した実施例１において、エージェント103aによるタスクの実行中に、エージェント103aの実行終了及び起動、即ち再起動が発生した場合に、タスクの実行を継続する機能を追加したものである。
【０１３５】
図１０（ａ）は、実施例４のタスクの再起動に対応したタスク起動処理の流れを示すフローチャートである。図１０（ａ）及び後述する図１０（ｂ）に示す処理は、監視装置103のＣＰＵ201が外部記憶装置207に格納されたプログラムをＲＡＭ202にロードして実行することにより実現されるエージェント103aにより実行される。
【０１３６】

実施例４のタスク起動処理において、エージェント103aのタスク実行部604は、S701のタスク実行クラスのインスタンスの作成後、S1001に処理を進める。
S1001において、タスク実行部604は、タスク起動処理に引数として渡された［タスクＩＤ、タスククラス名］の組を記憶装置に保存する。ここで記憶装置とは、例えばエージェント103aが内部に持っているデータベースや、タスクＩＤとタスククラス名を属性に持つデータのリストを表現したＪＳＯＮファイル等であり、その実態は例えば監視装置103の外部記憶装置207に記憶される。またS1001では保存に際して、同じタスクＩＤを持つ［タスクＩＤ，タスククラス名］の組が既に存在する場合は上書きするものとする。また、S1001では、タスクＩＤが空のＵＵＩＤ（全てが0のＵＵＩＤ）である場合、［タスクＩＤ，タスククラス名］の組は保存しないものとする。

【０１３７】
なお、図示しないが、各タスク実行クラスのOnSubtaskEnd関数は、タスクに紐づいた全てのサブタスクに対する処理が完了した場合、上記S1001で保存された［タスクＩＤ、タスククラス名］の組を記憶装置から削除する。

【０１３８】
図１０（ｂ）は、実施例４においてエージェント103aが起動時に呼び出すタスク再起動処理の流れを示すフローチャートである。
タスク再起動処理において、タスク実行部604は、S1010において、図１０（ａ）のS1001でタスク起動処理が保存した［タスクＩＤ、タスククラス名］の組の一覧を、記憶装置から取得する。

【０１３９】
次にS1011において、タスク実行部604は、上記S1010で取得した一覧から未処理の［タスクＩＤ、タスククラス名］の組を順に取り出し、S1012に処理を進める。
S1012において、タスク実行部604は、上記S1011で［タスクＩＤ、タスククラス名］の組の取り出しに成功したか否かを判定する。タスク実行部604は、上記S1011で［タスクＩＤ、タスククラス名］の組の取り出しに成功したと判定した場合（S1012でYesの場合）、S1013に処理を進める。

【０１４０】
S1013において、タスク実行部604は、図１０（ａ）に示すタスク起動処理を呼び出した後、S1011に遷移し、次の［タスクＩＤ、タスククラス名］の組に処理を移行する。

【０１４１】
また上記S1012にて、タスク実行部604は、上記S1011で［タスクＩＤ、タスククラス名］の組の取り出しに失敗したと判定した場合（S1012でNoの場合）、即ち全ての［タスクＩＤ、タスククラス名］の組に対する処理が終了した場合、タスク再起動処理を終了する。

【０１４２】
なお実施例４では、図１０（ａ）のタスク起動処理のS702で呼び出される、タスク実行用クラスのStart関数の処理（図７（ｂ）のタスク開始処理）のS711で、管理アプリケーション101aから取得するサブタスク情報は、サブタスク情報のステータスが未完了（実行待ちまたは実行中）であるサブタスク情報の一覧である。これにより、実行が完了したサブタスク情報に対する処理が再度実行されることが抑止され、実行が完了していないサブタスクが処理される。

【０１４３】
以上のように、上記各実施例によれば、ネットワーク機器の分散管理システムを、単一の管理サプリケーションと複数のエージェントとにより構成する。管理アプリケーションは各エージェントと、各エージェントは管理アプリケーションと各ネットワーク機器と通信する構成とする。複数のネットワーク機器に同種の処理を一斉に実行する際、管理アプリケーションが事前に各ネットワーク機器への処理情報をサブタスク情報に分解する。エージェントは各サブタスク情報に従って各ネットワーク機器に対する処理を実行し、実行結果を管理アプリケーションに送信する。管理アプリケーションは受信したサブタスクの実

行結果から生成したサブタスク情報で事前に生成したサブタスク情報を更新する。これにより、複数のネットワーク機器に同種の処理を実行するに際し、処理の内容に関係なく、管理アプリケーションとエージェント間の共通の呼び出しを使うことが可能となる。よって、管理アプリケーションとエージェント間の呼び出しの管理と管理アプリケーションおよびエージェント上の処理の実装が容易になる。また、処理に紐づいたサブタスクのステータスを調べることにより、処理の進行状況の把握が可能となる。また、各ネットワーク機器に対する処理全体をエージェントに移譲することが可能となり、管理アプリケーションの負荷を軽減することが可能となり、大量のネットワーク機器の管理が可能となる。従って、複数のエージェントを用いて分散管理する複数のデバイスに処理を実行させる場合でも、処理の内容に関係なく管理装置からエージェントに対してデバイスへの処理を指示でき、効率的にデバイスに対して処理を実行させることができる。

【０１４４】

なお、上述した各種データの構成及びその内容はこれに限定されるものではなく、用途や目的に応じて、様々な構成や内容で構成されていてもよい。

以上、一実施形態について示したが、本発明は、例えば、システム、装置、方法、プログラムもしくは記憶媒体等としての実施態様をとることが可能である。具体的には、複数の機器から構成されるシステムに適用しても良いし、また、一つの機器からなる装置に適用しても良い。

また、上記各実施例を組み合わせた構成も全て本発明に含まれるものである。

【０１４５】

（その他の実施例）

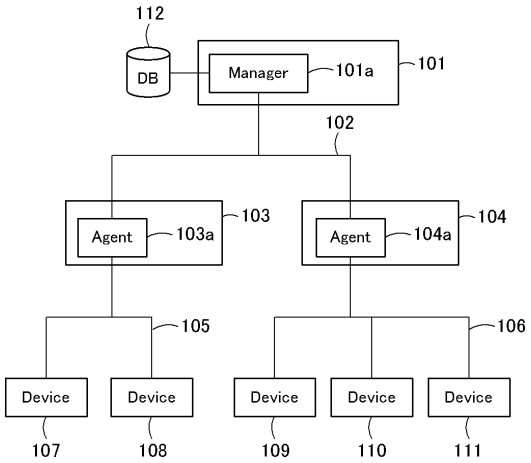
本発明は、上述の実施例の1以上の機能を実現するプログラムを、ネットワーク又は記憶媒体を介してシステム又は装置に供給し、そのシステム又は装置のコンピュータにおける１つ以上のプロセッサーがプログラムを読出し実行する処理でも実現可能である。また、1以上の機能を実現する回路（例えば、ＡＳＩＣ）によっても実現可能である。

また、本発明は、複数の機器から構成されるシステムに適用しても、１つの機器からなる装置に適用してもよい。

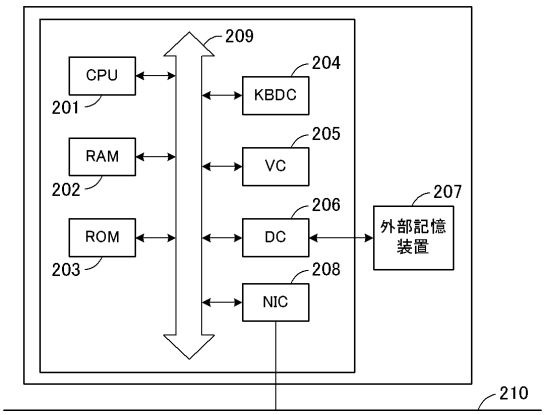
本発明は上記実施例に限定されるものではなく、本発明の趣旨に基づき種々の変形（各実施例の有機的な組合せを含む）が可能であり、それらを本発明の範囲から除外するものではない。即ち、上述した各実施例及びその変形例を組み合わせた構成も全て本発明に含まれるものである。

【図面】

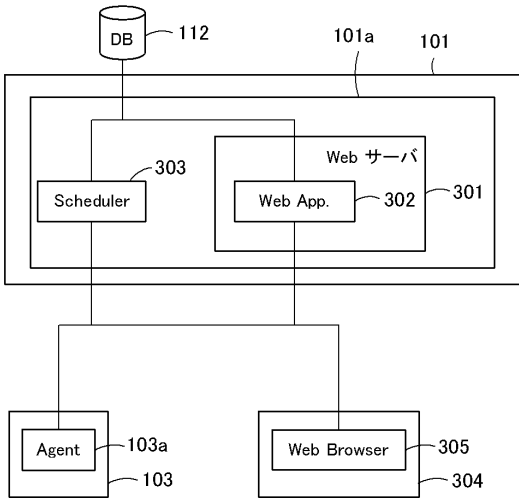
【図１】


112
DB
Manager
101a
101
102
Agent
103a
103
Agent
104a
104
105
106
Device
Device
Device
Device
Device
107
108
109
110
111

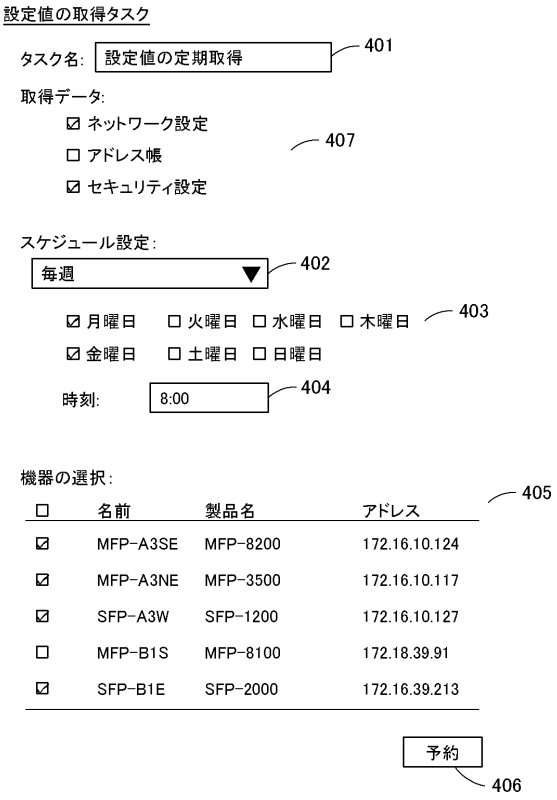
【図２】


209
CPU
201
RAM
202
ROM
203
204
KBDC
205
VC
206
DC
208
NIC
207
外部記憶装置
210

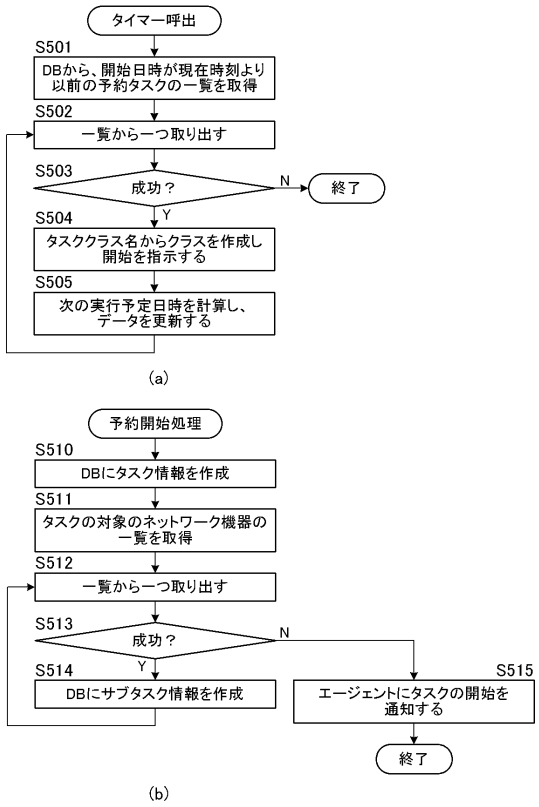
【図３】


DB
112
101a
101
Web サーバ
303
Scheduler
Web App.
302
301
Agent
103a
103
Web Browser
305
304

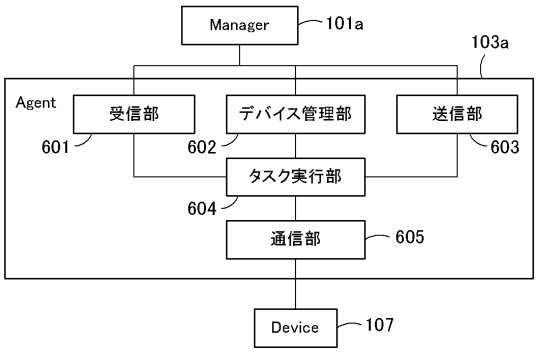
【図４】


設定値の取得タスク
タスク名:
設定値の定期取得
401
取得データ:
☑ ネットワーク設定
☐ アドレス帳
☑ セキュリティ設定
407
スケジュール設定:
毎週
402
☑ 月曜日 ☐ 火曜日 ☐ 水曜日 ☐ 木曜日
☑ 金曜日 ☐ 土曜日 ☐ 日曜日
403
時刻:
8:00
404
機器の選択:
405
☐ 名前 製品名 アドレス
☑ MFP-A3SE MFP-8200 172.16.10.124
☑ MFP-A3NE MFP-3500 172.16.10.117
☑ SFP-A3W SFP-1200 172.16.10.127
☐ MFP-B1S MFP-8100 172.18.39.91
☑ SFP-B1E SFP-2000 172.16.39.213
予約
406

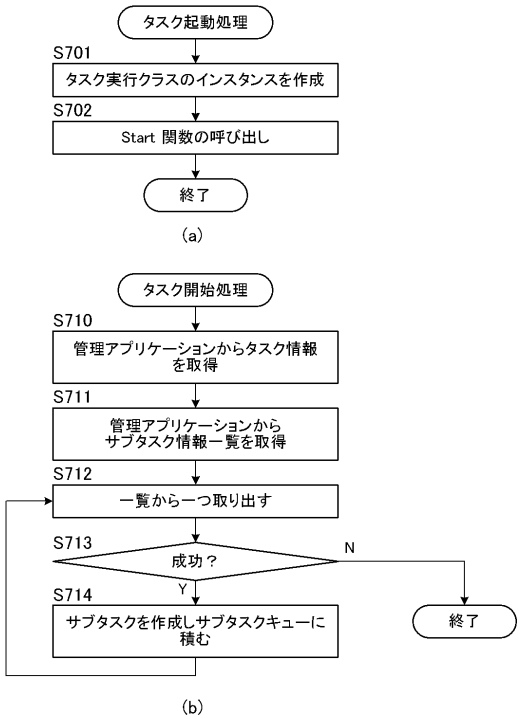
【図５】


タイマー呼出
S501
DBから、開始日時が現在時刻より以前の予約タスクの一覧を取得
S502
一覧から一つ取り出す
S503
成功？
N
終了
Y
S504
タスククラス名からクラスを作成し開始を指示する
S505
次の実行予定日時を計算し、データを更新する
(a)
予約開始処理
S510
DBにタスク情報を作成
S511
タスクの対象のネットワーク機器の一覧を取得
S512
一覧から一つ取り出す
S513
成功？
N
Y
S514
DBにサブタスク情報を作成
S515
エージェントにタスクの開始を通知する
終了
(b)

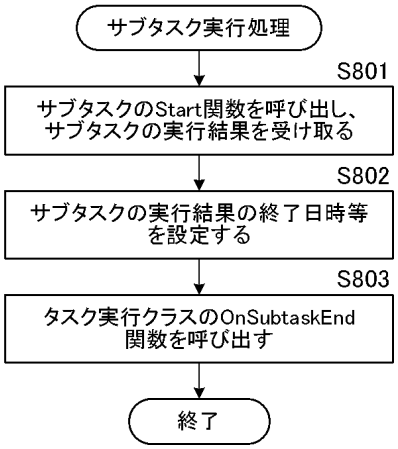
【図６】


Manager
101a
103a
Agent
受信部
デバイス管理部
送信部
601
602
603
タスク実行部
604
通信部
605
Device
107

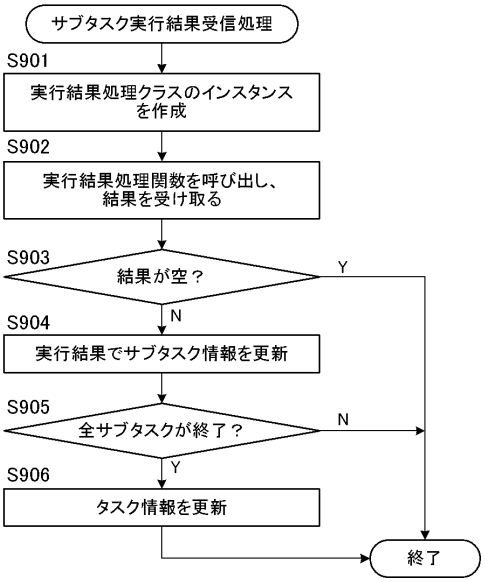
【図７】


タスク起動処理
S701
タスク実行クラスのインスタンスを作成
S702
Start 関数の呼び出し
終了
(a)
タスク開始処理
S710
管理アプリケーションからタスク情報を取得
S711
管理アプリケーションからサブタスク情報一覧を取得
S712
一覧から一つ取り出す
S713
成功？
N
Y
S714
サブタスクを作成しサブタスクキューに積む
終了
(b)

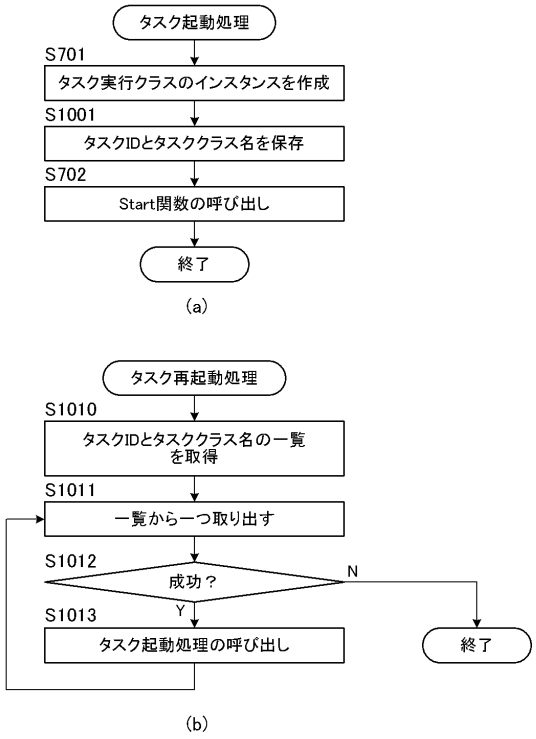
【図８】


サブタスク実行処理
S801
サブタスクのStart関数を呼び出し、サブタスクの実行結果を受け取る
S802
サブタスクの実行結果の終了日時等を設定する
S803
タスク実行クラスのOnSubtaskEnd関数を呼び出す
終了

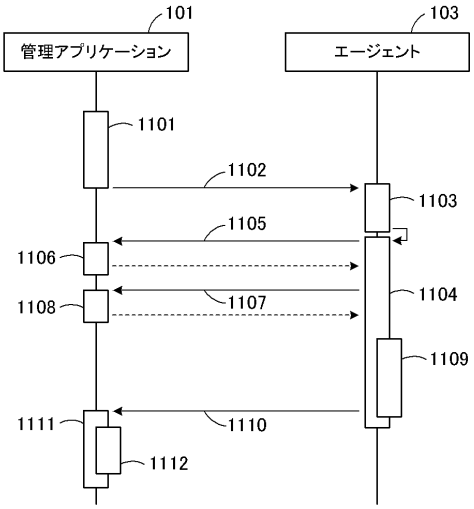
【図９】


サブタスク実行結果受信処理
S901
実行結果処理クラスのインスタンスを作成
S902
実行結果処理関数を呼び出し、結果を受け取る
S903
結果が空？
Y
N
S904
実行結果でサブタスク情報を更新
S905
全サブタスクが終了？
N
Y
S906
タスク情報を更新
終了

【図１０】


タスク起動処理
S701
タスク実行クラスのインスタンスを作成
S1001
タスクIDとタスククラス名を保存
S702
Start関数の呼び出し
終了
(a)
タスク再起動処理
S1010
タスクIDとタスククラス名の一覧を取得
S1011
一覧から一つ取り出す
S1012
成功？
N
Y
S1013
タスク起動処理の呼び出し
終了
(b)

【図１１】


101
管理アプリケーション
103
エージェント
1101
1102
1103
1105
1106
1104
1108
1107
1109
1111
1110
1112

フロントページの続き

(56)参考文献　　特開２０１５－１７６５９４（ＪＰ，Ａ）
国際公開第２０１４／１７４６７１（WO，Ａ１）

(58)調査した分野　(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　１３／００
Ｇ０６Ｆ　　９／４８
Ｇ０６Ｆ　１１／３４
Ｇ０６Ｆ　１１／３０