**REPÚBLICA FEDERATIVA DO BRASIL**
MINISTÉRIO DA INDÚSTRIA, COMÉRCIO EXTERIOR E SERVIÇOS
**INSTITUTO NACIONAL DA PROPRIEDADE INDUSTRIAL**

# CARTA PATENTE Nº PI 0209761-3

O INSTITUTO NACIONAL DA PROPRIEDADE INDUSTRIAL concede a presente PATENTE DE INVENÇÃO, que outorga ao seu titular a propriedade da invenção caracterizada neste título, em todo o território nacional, garantindo os direitos dela decorrentes, previstos na legislação em vigor.

**(21) Número do Depósito:** PI 0209761-3

**(22) Data do Depósito:** 30/05/2002

**(43) Data da Publicação do Pedido:** 05/12/2002

**(51) Classificação Internacional:** G06F 9/445

**(30) Prioridade Unionista:** US 60/294,331 de 30/05/2001

**(54) Título:** SISTEMA DE TEMPO DE EXECUÇÃO DE DISPOSITIVO DE COMUNICAÇÃO MÓVEL PARA EXECUTAR UMA APLICAÇÃO, MÉTODO DE MANUSEAR UM MÓDULO LIGADO AO PRINCIPAL EM UM SISTEMA ALVO E PRODUTO DE PROGRAMA DE COMPUTADOR

**(73) Titular:** BLACKBERRY LIMITED. Endereço: 2200 University Avenue E., Waterloo, Ontario, N2K 0A7, CANADÁ (CA)

**(72) Inventor:** DAVID O. YACH; JOHN F. A. DAHMS

**Prazo de Validade:** 10 (dez) anos contados a partir de 03/04/2018, observadas as condições legais

**Expedida em:** 03/04/2018

Assinado digitalmente por:
**Júlio César Castelo Branco Reis Moreira**
Diretor de Patente

# SISTEMA DE TEMPO DE EXECUÇÃO DE DISPOSITIVO DE COMUNICAÇÃO MÓVEL PARA EXECUTAR UMA APLICAÇÃO, MÉTODO DE MANUSEAR UM MÓDULO LIGADO AO PRINCIPAL EM UM SISTEMA ALVO E PRODUTO DE PROGRAMA DE COMPUTADOR

## APLICAÇÃO RELACIONADA

[001] Esta aplicação reivindica prioridade para o pedido provisório de Patente dos Estados número de série 60/294.331, intitulado "Method of Splitting A Processing Machine Runtime Between A Host System And a Target System" requerido em 30 de maio de 2001. Por esta referência, a revelação integral, incluindo os desenhos, do pedido provisório de Patente dos Estados Unidos número de série 60/294.331 foi aqui incorporado.

## FUNDAMENTOS DE INVENÇÃO

[002] Esta invenção relaciona-se ao campo de ambientes de tempo de execução de máquinas de processamento. Em particular, esta invenção relaciona-se a um método de subdivisão de um tempo de execução de máquina de processamento entre um sistema principal e um sistema alvo para conservar recursos no sistema alvo.

[003] Atualmente, a máquina virtual do estado da técnica é a máquina virtual Java™ (JVM) da Sun Microsystems, Inc. (Sun). No centro da tecnologia Java™ da Sun Microsystems está seu código de máquina virtual Java™, ou código de byte, conforme especificado atualmente pelo formato de arquivo de classe no capítulo 4 da segunda edição de The Java™ Virtual Machine Specification (Relatório sobre a máquina virtual Java™) por Tim Lindholm e Frank Yellin, Addison-Wesley Pub Co; ISBN; 0201432943.

[004] O código de byte de arquivo de classe coopera com

o Java™ Runtime Environment (JRE) da Sun, nas plataformas Solaris™, Win32, Linux™, Mac, e possivelmente em outras plataformas. Tipicamente, o código fonte escrito na linguagem de programação Java™ é compilado em código de byte de máquina virtual respeitando o formato de arquivo de classe ao utilizar um compilador Java™, como o "javac", e depois executado utilizando o JRE ou um ambiente de tempo de execução compatível e uma máquina de processamento.

[005] Com referência à Figura 1, um diagrama de blocos da arquitetura JRE em camadas ilustra vários aspectos da técnica Sun. Vários mecanismos (100A e 100B) fornecem arquivos de classe de código de byte do programa de software (110A e 110B). Por exemplo, o computador 100A compila o software 110A em arquivos de classe de código de byte. Alternativamente, um navegador (browser) da Web poderá utilizar um software "plugin" 110B para baixar os arquivos de classe de código de byte do software 100B.

[006] O código de byte em um arquivo de classe normalmente se refere a várias outras classes, cada uma das quais tem um arquivo de classe. Por essa razão, arquivos de classe de pacotes padrão 120 são fornecidos como um recurso de software compartilhado para serem reutilizados por instâncias do software (110A e 110B). A JVM 140 obtém arquivos de classe e executa o software (110A e 110B) e arquivos de classe de pacote padrão 120.

[007] Também estão mostrados os vários sistemas 130 em cima dos quais o JRE 142 opera. Os pacotes padrão em um tempo de execução definem uma plataforma de tempo de execução particular especificada na interface do programador de aplicação (API).

[008] A Java™ 2 Standard Edition (J2SE) é uma API da plataforma de referência Sun. Eles também fornecem uma implementação de referência que compreende um JRE configurado com um conjunto de pacotes padrão que processam na JVM. Os desenvolvedores de aplicação podem escrever aplicações na linguagem de programação Java™ que se refere às classes do pacote padrão J2SE e poderão esperar ter suas aplicações processando em sistemas de tempo de execução J2SE compatíveis. Existem outras plataformas que são normalmente definidas por comparação a J2SE. Por exemplo, um superconjunto da J2SE, o Java™ 2 Enterprise Edition (J2EE) acrescenta outras características. De interesse particular é um subconjunto do J2SE, a Java™ 2 Micro Edition (J2ME).

[009] Embora a plataforma J2SE poderá ser bem adequada para operar em sistemas como aqueles ilustrados pelas plataformas Solaris™, Win32, Linux™, e outros blocos 130 da Figura 1, o J2SE poderá não ser bem adaptado para operar em muitos dispositivos. Por exemplo, os arquivos de classe dos pacotes J2SE padrão podem atualmente consumir bem acima de 16 Megabytes de espaço em disco, que poderá superar a capacidade de armazenamento de muitos dispositivos.

[010] Para resolver este problema, Sun introduziu a plataforma Java™ 2 Micro Edition (J2ME), máquinas virtuais adicionais, e configurações de dispositivos associados.

[011] A Connected Limited Device Configuration (CLDC) e K Virtual Machine (KVM) são endereçadas a pequenos dispositivos de consumidor portáteis, com 128K a 512K de memória, e quando utilizado com o Mobile Information Device Profile (MIDP) poderá fornecer um ambiente de aplicação

para dispositivos como os telefones celulares e dispositivos bilaterais de radiochamada.

[012] A Connected Device Configuration (CDC) e a C Virtual Machine (CMV) são endereçados a dispositivos de consumidor de próxima geração emergentes com 2 MB de memória ou mais, e quando utilizados com o Foundation Profile (FP) poderão fornecer um ambiente de aplicação para os dispositivos de consumidor de próxima geração.

[013] Uma vantagem de J2ME é que quando ele é utilizado com as configurações CLDC ou CDC anteriormente mencionadas, menos pacotes de classe padrão são armazenados em muitos dispositivos quando comparado com o J2SE. Portanto, J2ME poderá tomar menos espaço em um dispositivo às custas de não suportar todas as características do J2SE.

[014] Embora a tecnologia de tempo de execução Java™ possa estar disponível para sistemas e dispositivos diferentes, e embora a plataforma J2ME resolva o problema de espaço de armazenamento limitado dos dispositivos ao remover a funcionalidade, J2ME poderá não ser considerado uma solução adequada, pois ele poderá não resolve a eficiência de uma implementação de tempo de execução do dispositivo. Assim, há uma necessidade de que o tempo de execução seja otimizado para um dispositivo alvo (bem como outras necessidades).

[015] Para melhor compreender a presente invenção, a informação seguinte sobre a tecnologia de tempo de execução da Java é fornecida. De acordo com Lindholm et al., na seção §2.17.1 do relatório de JVM da Sun: “A máquina virtual Java inicia a execução ao invocar o principal método de alguma classe especificada e passa para ele um

único argumento, que é um conjunto de caracteres. Isto faz com que a classe especificada seja carregada (§2.17.2), enlaçada (§2.17.3) a outros tipos que ela utiliza, e inicializada (§2.17.4)". Portanto, ao especificar o nome de uma classe "principal" quando do início da JVM 140 da Figura 1, um arquivo de classe será carregado e a execução das instruções de código de byte será inicializada no ponto de entrada principal estático daquele arquivo de classe. Ademais, tipos referenciados, como as classes, utilizadas pela classe "principal" serão enlaçados e inicializados. Dependendo da utilização de outras classes pelo arquivo de classe "principal", recursos de tempo de execução significativos serão consumidos para carregar e enlaçar os arquivos de classe utilizados.

[016] A tecnologia de tempo de execução Java™ exige que o sistema de tempo de execução carregue e enlace todos os arquivos de classe exigidos toda vez que uma classe "principal" for especificada para execução, podendo causar o consumo precipitado de recursos em um sistema alvo, tal como um dispositivo.

[017] Uma aplicação Java™ típica tem pelo menos um arquivo de classe "principal" contendo um ponto de entrada principal estático, possivelmente tendo ainda vários arquivos de classe de suporte.

[018] A seguinte listagem de programa Java™ exemplar é considerada a seguir:

```
public class Hello {
     public static void main(String  a){
          System.out.println("Hello!");
          Bye.bye(a);
```

```
        }
    }
    public class Bye{
        public static void bye(String  a){
            System.out.printin("Bye!");
        }
    }
```

[019] A listagem acima fornece código fonte para duas classes, Hello e Bye, cada uma das quais pode ser compilada em formato de arquivo de classe de uma maneira conhecida pelos versados na técnica, tal como através da colocação da fonte para cada classe em um arquivo Hello.java e um arquivo Bye.java e utilizar o comando "javac Hello.java Bye.java" para obter um arquivo Hello.class e um arquivo Bye.class.

[020] A classe Hello fornece um ponto de entrada principal estático e, portanto, é adequado para ser especificado como uma classe "principal" quando do início da JVM 140.

[021] Com referência à Figura 2, a técnica de enlace do tempo de execução da Figura 1 é considerada com referência ao programa "Hello" do exemplo acima. Uma pluralidade de arquivos de classe 200 está disponível para a máquina virtual 140 (da Figura 1). Cada arquivo de classe tem informação simbólica que é utilizada pela máquina virtual 140 para resolver referências a outros arquivos de classe utilizados.

[022] Tipicamente, o arquivo Hello.class 210A é carregado em 220A primeiro como ele é especificado quando iniciar a JVM 140. A JVM 140 então procede com a execução

das instruções de código de byte no ponto de entrada principal da classe carregada 220A. Como a classe Hello 220A utiliza várias classes de pacote padrão, os arquivos de classe para as classes utilizadas serão carregados e enlaçados para a classe Hello 220A. O arquivo Object.class 210B será carregado em 220B e enlaçado 230B para a classe Hello 210A. De modo similar, o arquivo String.class 210C, o arquivo System.class 210D, e outros arquivos de classe 210 utilizados pela classe Hello serão carregados em 220C, 220D, 220 e enlaçados em 230C, 230D e 230. A classe Hello também utiliza a classe Bye (como classe de suporte que não é uma classe de pacote padrão) de modo que o arquivo Bye.class 210E será carregado em 220E e enlaçado em 230E.

[023] Embora não seja expressamente mostrado nos desenhos, cada vez que um arquivo de classe 210 é carregado em 220 e enlaçado em 230, quaisquer arquivos de classes que a classe carregada 220 utiliza também poderão ser carregados e enlaçados. Por exemplo, no caso da classe Bye de suporte carregada 220E, ela utiliza muitas das mesmas classes que a classe Hello 210A. Dependendo de quando a classe Bye 220E for carregada e enlaçada 230E, a classe Bye 220E poderá não ter que carregar os arquivos de classe 210 que forem comuns às classes também utilizadas e carregadas pela classe Hello. No entanto, todas as classes utilizadas pelo Bye 220A em última instância terão de ser enlaçados ao Bye assim como para o Hello para serem capazes de utilizar a classe Bye de suporte. A situação é similar com as classes de pacote padrão.

[024] O carregamento (em 220) e o enlace (em 230) do arquivo de classe tradicional 210 consomem recursos de

tempo de execução significativos e podem desacelerar a execução de um programa “principal” 220A quando o carregamento e o enlace dos arquivos de classe é disparado pela especificação de um comando para executar um programa, como será discutido mais a seguir com referência às Figuras 3A e 3B.

[025] Com referência às Figuras 3A e 3B, é discutido um fluxograma que ainda ilustra a técnica de enlace de tempo de execução da Figura 2, particularmente ilustrando a resolução tardia opcional. A classe “principal” é carregada em 310 do armazenamento de classe 200, com um disco rígido ou uma rede. A classe é verificada e preparada em 315. Se a resolução tardia não é utilizada conforme determinado em 320, então todas as classes utilizadas são enlaçadas e carregadas em 325. Independentemente de se a resolução tardia é ou não utilizada em 320, a classe “principal” é inicializada em 330.

[026] Instruções do ponto de entrada principal são apanhadas em 335. Se a instrução apanhada não envolve uma referência não resolvida conforme determinado em 340, a instrução apanhada é executada em 345. No entanto, se a instrução apanhada envolve uma referência de identificador não resolvida conforme determinado em 340, como a referência de classe para uma classe que ainda não foi carregada, então se a resolução tardia não é utilizada conforme determinado em 350, uma exceção é jogada no tempo de execução. Se a resolução tardia é utilizada conforme determinado em 350, e se a classe referenciada não puder ser carregada em 355, uma exceção é jogada no tempo de execução. Entretanto, se a resolução tardia é utilizada

conforme determinado em 350, e a classe referenciada pode ser carregada, a classe referenciada é carregada e a referência é resolvida em 360 antes da execução da instrução em 345. Se há mais instruções a executar conforme determinado em 365, então a instrução seguinte é apanhada em 335 ou então a máquina virtual encerra.

[027] Se a resolução tardia foi utilizada, então vários arquivos de classe teriam sido carregados e enlaçados em 360 durante a execução do código de byte do programa principal. Alternativamente, se a resolução tardia não foi utilizada, vários arquivos de classe teriam sido carregados e enlaçados em 325 antes da execução do código de byte do programa principal, após especificar o arquivo de classe “principal” para a JVM 140. Em qualquer caso, um consumo precipitado de recursos para carregar e enlaçar poderá ocorrer entre o tempo em que o programa principal foi especificado para execução pela JVM 140 no tempo de execução e o tempo em que o programa principal quer terminou ou jogou uma exceção.

[028] Portanto, mesmo ao eliminar a resolução tardia, há um risco potencial para o consumo precipitado de recursos para carregar e enlaçar arquivos de classe em execução disparada de enlace e de carregamento.

**SUMÁRIO DA INVENÇÃO**

[029] A presente invenção supera os problemas observados acima bem como outros. De acordo com os ensinamentos da presente invenção, um sistema e um método são fornecidos para classes de pré-enlace para uso por uma ou mais aplicações. O sistema e o método também poderão ser utilizados quando o processamento do tempo de execução for

dividido entre um sistema principal e um sistema alvo. No sistema principal pelo menos várias classes são carregadas e enlaçadas. Pelo menos um módulo enlaçado no principal é gerado das classes enlaçadas. O módulo enlaçado no principal é tornado disponível para uso por uma ou mais aplicações que operam no sistema alvo.

**BREVE DESCRIÇÃO DOS DESENHOS**

[030] Para que a invenção possa ser mais claramente compreendida, versões da mesma serão descritas agora em detalhe por meio apenas de exemplo, com referência aos desenhos acompanhantes, nos quais:

A Figura 1 é um diagrama de blocos que ilustra uma técnica de tempo de execução do estado da técnica;

A Figura 2 é um diagrama de blocos que ilustra uma técnica de enlace do tempo de execução da Figura 1;

As Figuras 3A e 3B são fluxogramas que ainda ilustram a técnica de enlace de tempo de execução da Figura 2, particularmente ilustrando a resolução tardia opcional;

As Figuras 4 e 5 são diagramas de blocos que ilustram um sistema de tempo de execução dividido exemplar;

As Figuras 6 e 7 são diagramas de blocos que ilustram diferentes sistemas exemplares de tempo de execução divididos diferentemente;

A Figura 8 é um diagrama de blocos que ilustra uma técnica de enlace para um sistema de tempo de execução dividido;

A Figura 9 é um fluxograma que ainda ilustra a técnica de enlace da Figura 8, introduzindo uma etapa de enlace do principal e uma etapa de enlace do alvo;

A Figura 10 é um fluxograma que ainda ilustra a etapa

de enlace do principal da Figura 9; e

A Figura 11 é um fluxograma que ainda ilustra a etapa de enlace do alvo da Figura 9.

[031] Os mesmos números de referência são utilizados em figuras diferentes para referirem-se a elementos similares.

**DESCRIÇÃO DETALHADA DA INVENÇÃO**

[032] A Figura 4 mostra uma versão de um sistema de tempo de execução com base em módulo. Em vez de arquivos de classe não enlaçados, uma máquina de processamento 440 executa módulos 450 que incluem classes que já foram carregadas e enlaçadas. Mais especificamente, os módulos 450 compreendem a informação encontrada em arquivos de classe de conjunto fechado carregados e enlaçados, otimizando assim comandos, informação simbólica, tamanho do código e velocidade para a máquina de processamento alvo 440. Os módulos 450 permitem que o tempo de execução reutilize o trabalho intermediário de carregamento e de enlace em múltiplas execuções de programas principais, em vez de repetir este trabalho em cada execução. Os módulos 450 fornecem uma alternativa ao carregamento e enlace disparada pela execução.

[033] Um compilador (ou outro mecanismo) 405 recebe um arquivo de classe 407 que inclui referências simbólicas 409 a outras classes 411. O compilador 405 processa os arquivos de classe 407 e 411 que estão em código de byte tal que as referências simbólicas 409 são resolvidas. Os arquivos de classe processados são fornecidos para a máquina de processamento 440 como módulos 450. A máquina de processamento 440 opera mais eficientemente em dispositivos alvo 430 pois tipicamente o tamanho do módulo é

substancialmente menor que os arquivos de classe de tempo de execução tradicionais, por exemplo, poderá haver uma redução de aproximadamente oito vezes em comparação com o tamanho do arquivo de classe Java. Além disso, o código do módulo pode ser verificado uma vez utilizando verificações de sanidade antes de execuções múltiplas, aumentando assim as velocidades de execução subseqüentes. Os módulos podem ser configurados para minimizar a comunicação de código, particularmente útil nos dispositivos de comunicação de largura de banda limitada. Os módulos 450 podem ser configurados para minimizar a montagem do código e o tempo de execução, particularmente útil nos dispositivos de tempo de execução de recursos limitados. Os módulos 450 podem ser adaptados aos tempos de execução da máquina de processamento existente enquanto mantém compatibilidade com APIs de referência, como ilustra a Figura 5.

[034] A Figura 5 ilustra uma versão em que vários mecanismos 405A e 405B fornecem software 410. Um compilador 405A compila software 410. Alternativamente, outros mecanismos 405B podem ser utilizados para baixar ou de outra forma obter software 410. Pacotes de classe padrão 420 são fornecidos como recurso de software compartilhado a serem reutilizados por instâncias do software 410. A máquina de processamento 440 obtém classes e executa software 410 utilizando pacotes de classe padrão 420.

[035] Também são mostrados vários dispositivos alvo 430 no topo dos quais a máquina de processamento 440 opera, como o dispositivo móvel 430A, um assistente de dados pessoal (PDA) 430B, um aparelho 430C, um “thin client” 430D, ou outro dispositivo 430E.

[036] Na Figura 5, os módulos 450 foram introduzidos entre o mecanismo que fornece software 410 e a máquina de processamento 440 que executa o código de máquina. No entanto, a máquina de processamento 440 ainda utiliza as classes no software fornecido 410 bem como os pacotes de classe padrão 420, exceto que isto é agora feito através da utilização de módulos 450 em vez de diretamente utilizando arquivos de classe. Arquivos de classe ainda podem ser utilizados visto que o compilador pode aceitar tanto arquivos de classe como arquivos fonte em sua entrada e produzir módulos em sua saída.

[037] Por causa da presença de módulos 450, a máquina de processamento 440 não precisa ser uma máquina virtual nem mesmo conhecer sobre o formato de arquivo de classe, otimizando assim o desempenho no sistema alvo ao eliminar a necessidade de carregar, enlaçar, e resolver arquivos de classe. Outras otimizações são possíveis se o tempo de execução fornecido for dividido entre um sistema principal e um dispositivo alvo 430, conforme é descrito com referência à Figura 6.

[038] A Figura 6 ilustra uma versão dividida de um sistema de tempo de execução com base em módulo. O processamento do tempo de execução de classe é dividido entre um sistema principal 530 e um sistema alvo 430. A utilização de módulos permitem que o tempo de execução seja dividido eficientemente entre o sistema principal e o dispositivo alvo para otimizar a eficiência do tempo de execução no dispositivo alvo.

[039] No tempo de execução dividido do sistema principal, os arquivos de classe (407 e 411) são enlaçados

pelo principal em 510 dentro de módulos enlaçados pelo principal 520. O trabalho da análise do conjunto de arquivo de classe fechado é descarregado para o sistema principal 530. No tempo de execução dividido do sistema alvo, os módulos enlaçados pelo principal 520 são comunicados em 540 a partir do sistema principal 530, para serem enlaçados pelo alvo em 550 dentro de módulos enlaçados pelo alvo 560. Se qualquer resolução de classe adicional for necessária no sistema alvo 430, então os identificadores do módulo alvo adicionalmente necessários 562 são enlaçados pelo alvo em 550 com os módulos enlaçados pelo principal 520 para formar os módulos enlaçados pelo alvo 560. A máquina de processamento 440 executa os módulos enlaçados pelo alvo 560.

[040] A Figura 7 é um diagrama de blocos que ilustra outro sistema de tempo de execução dividido. Para o sistema principal 530, arquivos de classe 410 são enlaçados pelo principal em 510 dentro de módulos enlaçados pelo principal 520. Para este sistema, os detalhes do enlace pelo principal serão discutidas posteriormente com referência às Figuras 8, 9 e 10. Para o sistema alvo 430, os módulos enlaçados pelo principal 520 são comunicados em 540 do sistema principal 530, a serem enlaçados pelo alvo 550 em módulos enlaçados pelo alvo 560. A comunicação 540 entre o principal e o alvo poderá ocorrer sobre qualquer meio de modo que os módulos poderão ser fornecidos ao alvo, tal como através de uma rede de comunicação móvel se o alvo for um dispositivo de comunicação móvel, ou através de um sinal de dados incorporado em um sinal de portadora. Os detalhes do enlace pelo alvo serão discutidos posteriormente com

referência às Figuras 8, 9 e 11, abaixo.

[041] Com referência à Figura 8, um diagrama de blocos que ilustra o carregamento de arquivos de classe, e o enlace dividido em módulos, e a execução do tempo de execução de módulos divididos das Figuras 6 e 7 é descrito na mesma. O trabalho da análise do conjunto de arquivos de classe fechados é descarregado em um sistema principal 400A. Lá, os arquivos de classe 600 são carregados em 610 e enlaçados dentro de módulos enlaçados pelo principal 520A e 520F. São ilustrados um módulo de aplicação para o módulo do exemplo “Hello” 520A que compreende a informação otimizada encontrada no arquivo Hello.class 610A e o arquivo Bye.class 610E em que a classe Hello é pré-enlaçada à classe Bye. O módulo 520A também tem uma referência simbólica 615 a um módulo de Biblioteca 520F que compreende todas as classes do pacote padrão que as classes Hello e Bye utilizam, tais como as classes fornecidas pelo arquivo de classe Object 610B, pelo arquivo de classe String 610C, pelo arquivo de classe System 610D e outros arquivos de classe 610. O módulo Biblioteca 520F poderia exportar todos os símbolos públicos de modo que muitas classes “principais” diferentes como a Hello podem reutilizar a totalidade dos arquivos de pacote de classe padrão. Alternativamente, o módulo Biblioteca poderia compreender apenas aqueles arquivos de classe utilizados pelas classes Hello e Bye, ou mesmo todas as classes utilizadas poderiam ser incluídas diretamente no módulo 520A. O último caso eliminaria a necessidade de qualquer resolução de símbolos no sistema alvo.

[042] Quando pelo menos um módulo enlaçado pelo

principal (520A e/ou 520F) está disponível, é possível comunicar em 540 o módulo candidato 620A e 620F para o tempo de execução dividido no sistema alvo 400B. Uma vez que os módulos candidatos (620A e 602F) são recebidos no sistema alvo, eles são enlaçados pelo alvo em um módulo enlaçado pelo alvo 560A e 560F e quaisquer referências simbólicas de módulo 615 são resolvidas como é mostrado em 630. Uma classe de módulo principal pode ser especificada para execução, tal como a classe Hello 640. No entanto, vantajosamente, cada vez que o programa principal da classe Hello executa, não há nenhuma necessidade de resolver a referência 650 pois o enlace pelo alvo 630 o fornece.

[043] Com referência à Figura 9, é descrito um fluxograma que ilustra ainda a técnica de enlace da Figura 8, mostrando a etapa de enlace pelo principal e a etapa de enlace pelo alvo. No sistema de tempo de execução dividido no principal, as classes 600 são carregadas e enlaçadas pelo principal em 800 dentro de módulos enlaçados pelo principal 520. Então, pelo menos um módulo enlaçado pelo principal 520 é enviado para o sistema de tempo de execução dividido no alvo.

[044] No sistema de tempo de execução dividido no alvo, pelo menos um módulo enlaçado pelo principal candidato 620 é recebido em 720 do principal. O módulo enlaçado pelo principal candidato 620 é enlaçado pelo alvo em 900 em um módulo enlaçado pelo alvo 560. Pelo menos um módulo enlaçado pelo alvo 560 é executado em 730. Se um novo módulo for desejado conforme determinado em 740, o ciclo do processo de enlace pelo principal 800, processos de comunicação (710 e 720) e processo de enlace de alvo 900

poderão surgir. No entanto, se nenhum módulo novo for desejado, então a execução repetida em 730 dos módulos enlaçados pelo alvo pode surgir sem a despesa de carregar e enlaçar.

[045] Com referência à Figura 10, um fluxograma que ilustra ainda a etapa de enlace pelo principal da Figura 9 é descrito. No sistema de tempo de execução dividido no principal, os símbolos exportados pelos módulos enlaçados pelo principal 520 fornecem em 810 identificadores de módulo exteriores 815. Outrossim, as classes 600 fornecem em 820 classes principais candidatas 825. As referências de classe nas classes principais candidatas 825 são substituídas em 830 por referências de módulo utilizando os identificadores de módulos exteriores 815, assim fornecendo um conjunto fechado de classes de módulo candidato 835. Então, os identificadores exportados de módulo candidato 845 são fornecidos em 840. As classes de módulo candidato 835 e os identificadores exportados 845 são então verificados em 850. Se verificados conforme determinado em 860, então o módulo enlaçado pelo principal candidato é fornecido em 870 como um módulo enlaçado pelo principal 520. Se não for verificado conforme determinado em 860, uma exceção é jogada.

[046] Com referência à Figura 11, é descrito um fluxograma que ainda ilustra a etapa de enlace pelo alvo da Figura 9. No sistema de tempo de execução dividido no dispositivo, o módulo candidato recebido 620 fornece em 910 referências de módulo candidato 915. Outrossim, os módulos enlaçados pelo alvo 560 fornecem em 920 identificadores de módulo alvo 925. A seguir, a resolução das referências de

módulo no módulo candidato fornece em 930 um módulo resolvido 935. O módulo resolvido 935 é verificado em 940, e se o módulo resolvido 935 é verificado com sucesso conforme determinado em 950, então o módulo resolvido 935 é armazenado em 960 com outros módulos enlaçados pelo alvo 560. No entanto, se o módulo resolvido 935 não é verificado com sucesso conforme determinado por 950, uma exceção é jogada.

[047] Tendo descrito em detalhe as versões preferidas da presente invenção, incluindo modos preferidos de operação, deve ser compreendido que esta invenção e operação poderia ser construída e efetuada com elementos e etapas diferentes. As versões são apresentadas apenas por meio de exemplo e não têm o sentido de limitar o escopo do sistema e do método da presente invenção, que é definido pelas reivindicações.

[048] Para ilustrar o amplo escopo do sistema e do método, o que segue é fornecido. Código de máquina virtual é normalmente interpretado por software. No entanto, um processador de código de máquina virtual pode ser implementado em hardware. A adaptação do sistema e do método para um tempo de execução de máquina de processamento de hardware está dentro do escopo da invenção. Como exemplos adicionais do amplo escopo do sistema e do método, o sistema e o método poderão permitir a otimização de comandos, de informação simbólica e de código através da utilização dos módulos do sistema e do método. O sistema e o método poderão permitir um tamanho de módulo que é substancialmente menor que o arquivo de classe de tempo de execução tradicional, por exemplo ao reduzir em

alguns casos por oito vezes o tamanho do módulo em comparação com o tamanho do arquivo de classe Java sem perder a funcionalidade. O sistema e o método poderão prover que o código do módulo pode ser verificado utilizando verificações de sanidade uma vez antes de execuções múltiplas.

[049] O sistema e o método também poderão permitir que o módulo combine apenas novas classes para minimizar o tamanho do armazenamento do módulo e permitir a entrega eficiente do módulo para dispositivos de comunicação de largura de banda limitada. Como outro exemplo do amplo escopo do sistema e método, o sistema e o método podem combinar todas as classes necessárias para minimizar estabelecimento do código e tempo de execução em dispositivos de recursos limitados. O sistema e o método poderão ser adaptados aos tempos de execução de máquinas de processamento existentes enquanto mantém a compatibilidade com APIs de referência. O sistema e o método poderão prover que os módulos podem ser enlaçados dinamicamente enquanto minimiza o número de referências simbólicas. Ainda mais, o sistema e o método poderão prover que o código nos módulos pode ser executado diretamente do armazenamento em um dispositivo, diferente dos arquivos de classe que precisam ser carregados do armazenamento antes da execução.

**REIVINDICAÇÕES**

1. Sistema de tempo de execução de dispositivo de comunicação móvel para executar uma aplicação, compreendendo:

meios adaptados para receber um módulo pré-ligado (450, 520) que inclui classes (407, 411) que foram carregadas e ligadas,

o dito módulo pré-ligado (450, 520) tendo informação de arquivos de classe já carregados e ligados, em que um sistema principal (530) gera o módulo pré-ligado (450, 520);

meios (440) adaptados para executar uma aplicação operativa no dispositivo de comunicação móvel, em que a aplicação utiliza as classes (407, 411) durante a execução da aplicação,

a dita aplicação tendo acesso ao módulo pré-ligado (450, 520) durante a execução da aplicação, em que a necessidade da aplicação repetir o carregamento e a ligação das classes (407, 411) é eliminada durante a execução da aplicação devido à utilização do módulo pré-ligado (450, 520) incluindo classes (407, 411) que foram carregadas e ligadas,

**caracterizado** pelo fato de que os meios de recebimento são adaptados para receber o módulo pré-ligado (450, 520) através de um meio de comunicação a partir do sistema principal (530) durante o tempo de execução da aplicação, onde o meio de comunicação é uma rede de comunicações móveis através da qual o módulo pré-ligado (450, 520) é fornecido a um sistema alvo (430) que é um dispositivo de comunicação móvel, em que a aplicação opera sobre um

sistema alvo (430).

2. Sistema, de acordo com a reivindicação 1, **caracterizado** pelo fato do sistema alvo (430) incluir meios de ligação de alvo (550) para ligar o alvo do módulo pré-ligado (450, 520) a outros módulos para utilização pela aplicação.

3. Sistema, de acordo com a reivindicação 1, **caracterizado** pelo fato do módulo pré-ligado (450, 520) otimizar comandos, informação simbólica, e código.

4. Sistema, de acordo com a reivindicação 1, **caracterizado** pelo fato dos arquivos de classe conterem as classes e compreenderem informação simbólica.

5. Sistema, de acordo com a reivindicação 1, **caracterizado** pelo fato do módulo pré-ligado (450, 520) ser verificado.

6. Sistema, de acordo com a reivindicação 1, **caracterizado** pelo fato do módulo pré-ligado (450, 520) compreender uma pluralidade de módulos que contêm todas as classes necessitadas pela aplicação.

7. Sistema, de acordo com a reivindicação 1, **caracterizado** pelo fato do módulo pré-ligado (450, 520) compreender uma pluralidade de módulos, os módulos sendo ligados dinamicamente enquanto minimiza o número de referências simbólicas.

8. Sistema, de acordo com a reivindicação 1, **caracterizado** pelo fato da aplicação ser operativa em um dispositivo, em que o módulo pré-ligado (450, 520) é executável diretamente a partir do armazenamento no dispositivo.

9. Sistema, de acordo com a reivindicação 8,

**caracterizado** pelo fato do módulo pré-ligado (450, 520) fornecer a reutilização das etapas de carregamento e de ligação nas execuções de múltiplas aplicações que operam no dispositivo.

10. Sistema, de acordo com a reivindicação 1, **caracterizado** pelo fato do módulo pré-ligado (450, 520) ser gerado antes da execução da aplicação.

11. Sistema, de acordo com a reivindicação 1, **caracterizado** pelo fato do módulo pré-ligado (450, 520) ser configurado com base no estabelecimento do código e no tempo de execução.

12. Sistema, de acordo com a reivindicação 1, **caracterizado** pelo fato da aplicação operar sobre um dispositivo de dados móvel.

13. Sistema, de acordo com a reivindicação 1, **caracterizado** pelo fato da aplicação operar sobre um assistente de dados pessoal.

14. Sistema, de acordo com a reivindicação 1, **caracterizado** pelo fato da aplicação operar sobre um aparelho.

15. Sistema, de acordo com a reivindicação 1, **caracterizado** pelo fato da aplicação operar sobre uma aplicação de cliente fino.

16. Sistema, de acordo com a reivindicação 1, **caracterizado** pelo fato das classes compreenderem classes com base em Java.

17. Método de manusear um módulo ligado ao principal (520) em um sistema alvo (430), o módulo ligado ao principal (520) tendo sido gerado por um sistema principal (530) através do carregamento de pelo menos várias classes

(407, 411), o sistema principal (530) ligando no sistema principal (530) as classes (407, 411) carregadas e formando pelo menos um módulo ligado ao principal (520) a partir das classes (407, 411) ligadas, o método compreendendo:

receber o módulo ligado ao principal (520) no sistema alvo (430);

determinar se quaisquer classes adicionais devem ser ligadas ao módulo ligado ao principal (520),

se classes adicionais devem ser ligadas, então ligar o módulo ligado ao principal (520) às classes adicionais para formar um módulo ligado ao alvo (560); e

permitir que o módulo ligado ao alvo (560) seja utilizado por uma aplicação que opera no sistema alvo (430),

**caracterizado** pelo fato de o módulo ligado ao principal (520) ser recebido a partir do sistema principal (530) através de um meio de comunicação durante o tempo de execução da aplicação, em que o meio de comunicação é uma rede de comunicações móveis e o sistema alvo (430) é um dispositivo de comunicação móvel.

18. Método, de acordo com a reivindicação 17, **caracterizado** pelo fato do módulo ligado ao principal (520) ser fornecido ao sistema alvo (430) eliminando o sistema alvo (430) de resolver classes associadas ao módulo ligado ao principal (520), porque o módulo ligado ao principal (520) foi formado através da ligação das classes (407, 411) carregadas no sistema principal (530).

19. Método, de acordo com a reivindicação 17, **caracterizado** pelo fato do módulo ligado ao principal (520) ser fornecido para o sistema alvo (430) eliminando a

necessidade do sistema alvo (430) resolver classes associadas ao módulo ligado ao principal (520) porque o módulo ligado ao principal (520) foi formado através da ligação das classes (407, 411) carregadas no sistema principal (530).

20. Método, de acordo com a reivindicação 17, **caracterizado** pelo fato do sistema alvo (430) compreender uma máquina de processamento (440) para a execução da aplicação.

21. Método, de acordo com a reivindicação 17, **caracterizado** pelo fato do sistema alvo (430) compreender uma máquina virtual para a execução da aplicação.

22. Método, de acordo com a reivindicação 17, **caracterizado** pelo fato do sistema alvo (430) compreender um dispositivo de dados móvel.

23. Método, de acordo com a reivindicação 17, **caracterizado** pelo fato do sistema alvo (430) compreender um assistente de dados pessoal.

24. Método, de acordo com a reivindicação 17, **caracterizado** pelo fato do sistema alvo (430) compreender uma aplicação de cliente fino.

25. Método, de acordo com a reivindicação 17, **caracterizado** pelo fato do sistema alvo (430) compreender um processador de código de máquina virtual para processar o módulo ligado ao principal (520).

26. Método, de acordo com a reivindicação 17, **caracterizado** pelo fato do sistema maior compreender um tempo de execução de máquina de processamento de hardware para processar o módulo ligado ao principal (520).

27. Método, de acordo com a reivindicação 17,

**caracterizado** pelo fato das classes (407, 411) compreenderem classes com base em Java.

28. Método, de acordo com a reivindicação 17, **caracterizado** pelo fato do módulo ligado ao principal (520) ter sido verificado no sistema principal (530) utilizando verificações de sanidade uma vez antes das múltiplas execuções no sistema alvo (430).

29. Método, de acordo com a reivindicação 17, **caracterizado** pelo fato do módulo ligado ao principal (520) compreender uma pluralidade de módulos que contêm todas as classes necessitadas pela aplicação.

30. Método, de acordo com a reivindicação 17, **caracterizado** pelo fato do módulo ligado ao principal (520) compreender uma pluralidade de módulos, os módulos tendo sido ligados dinamicamente enquanto minimizam o número de referências simbólicas.

31. Método, de acordo com a reivindicação 17, **caracterizado** pelo fato do módulo ligado ao principal (520) ser executável diretamente a partir do armazenamento no sistema alvo (430).

32. Método, de acordo com a reivindicação 17, **caracterizado** pelo fato do módulo ligado ao principal (520) fornecer a reutilização das etapas de carregamento e de ligação em múltiplas execuções da aplicação.

33. Método, de acordo com a reivindicação 32, **caracterizado** pelo fato do sistema principal (530) resolver referências de módulo durante a geração do módulo ligado ao principal (520).

34. Método, de acordo com a reivindicação 17, **caracterizado** pelo fato do módulo ligado ao principal (520)

fornecer a reutilização das etapas de carregamento e de ligação em múltiplas execuções da aplicação e de uma segunda aplicação.

35. Método, de acordo com a reivindicação 17, **caracterizado** pelo fato do módulo ligado ao principal (520) ser gerado antes da execução da aplicação.

36. Método, de acordo com a reivindicação 17, **caracterizado** pelo fato do módulo ligado ao principal (520) ser configurado com base em comunicação por código ao sistema alvo (430).

37. Método, de acordo com a reivindicação 17, **caracterizado** pelo fato do módulo ligado ao principal (520) ser configurado com base no estabelecimento de código e no tempo de execução.

38. Método, de acordo com a reivindicação 17, **caracterizado** por compreender ainda a etapa de:

dividir o tempo de execução da máquina de processamento entre o sistema principal (530) e o sistema alvo (430) ao formar o módulo ligado ao principal (520) no sistema principal (530) e ao ligar ao alvo no sistema alvo (430) o módulo ligado ao principal (520) dentro de pelo menos um módulo ligado ao alvo (560) para utilização pela aplicação.

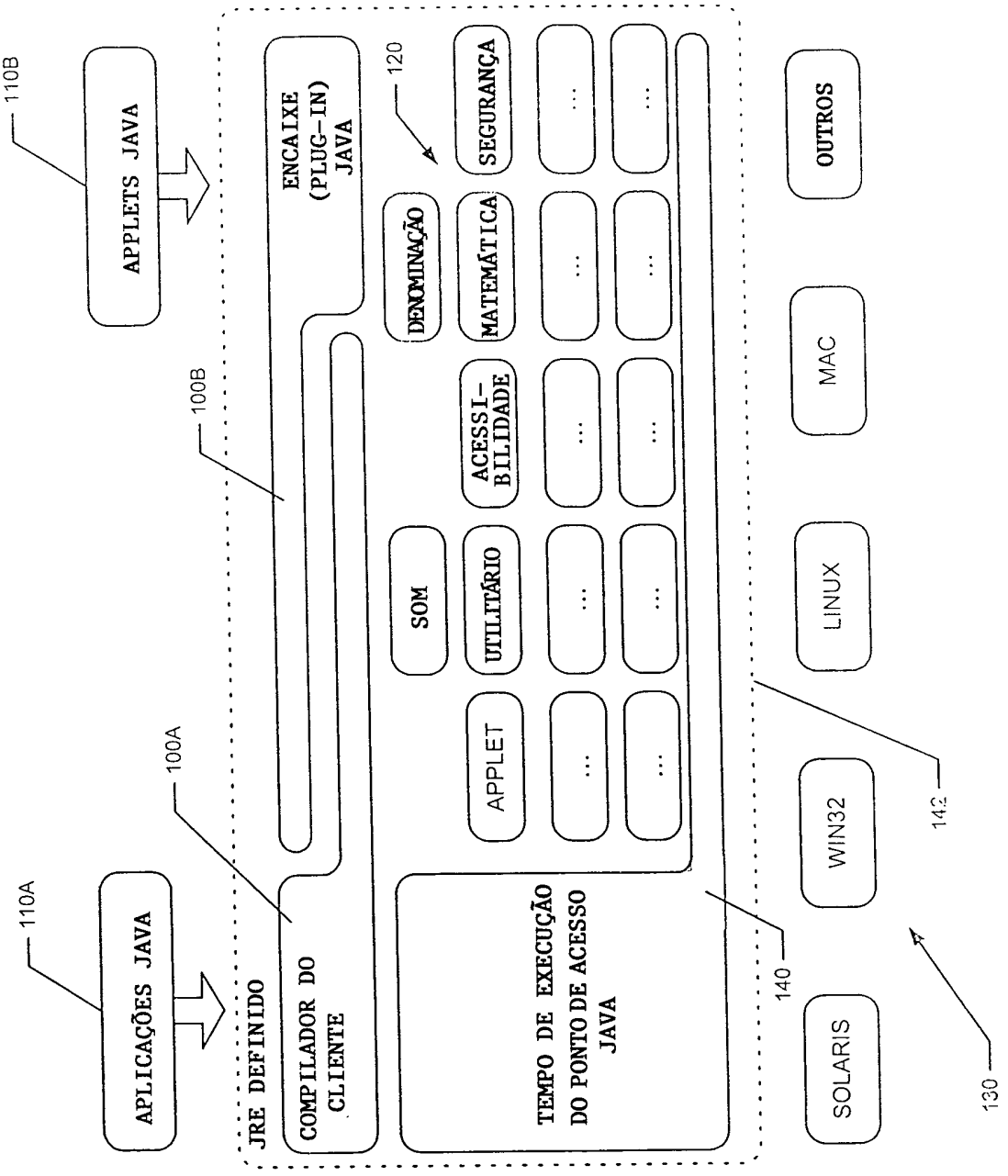


Fig. 1

ESTADO DA TÉCNICA

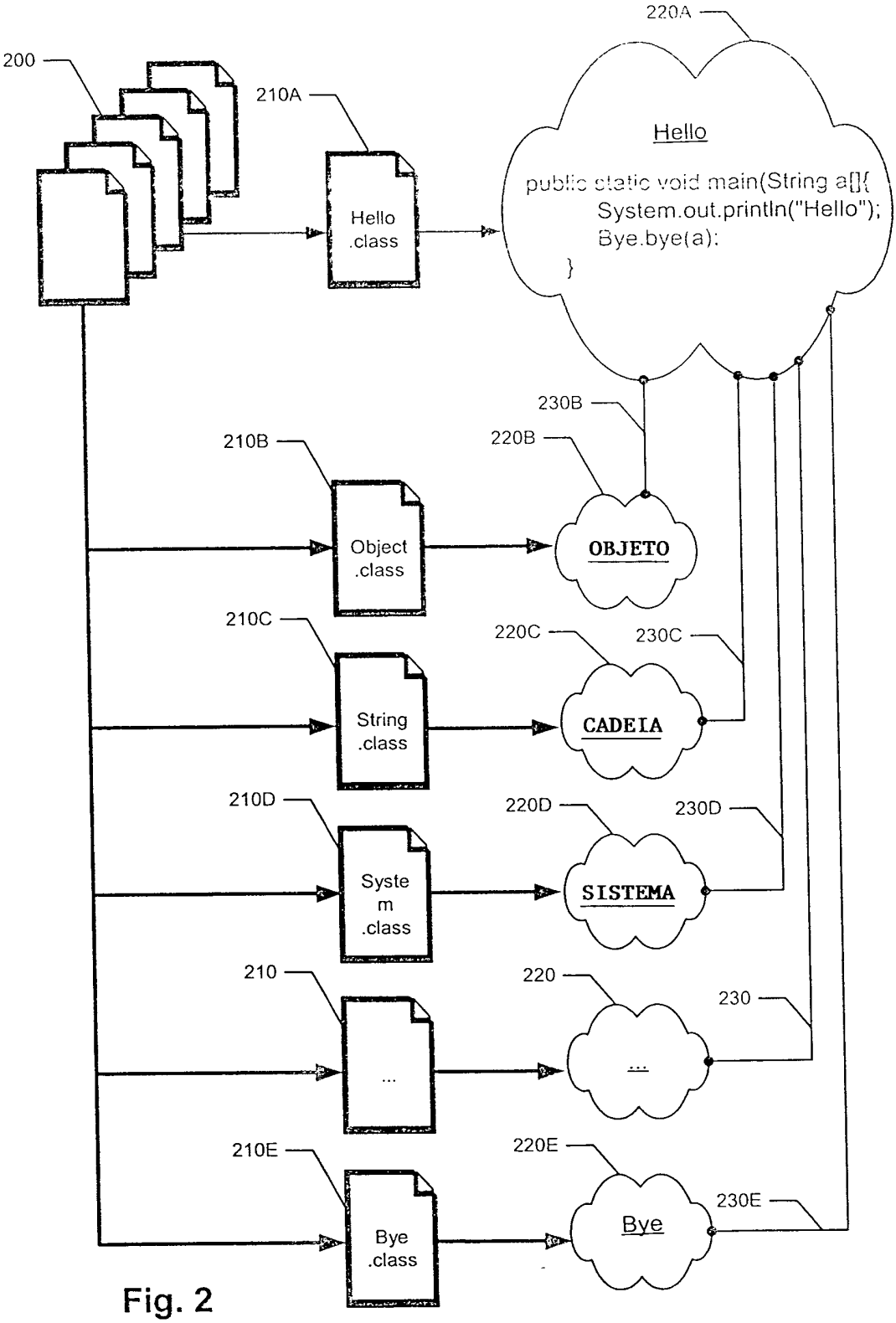


Fig. 2

ESTADO DA TÉCNICA

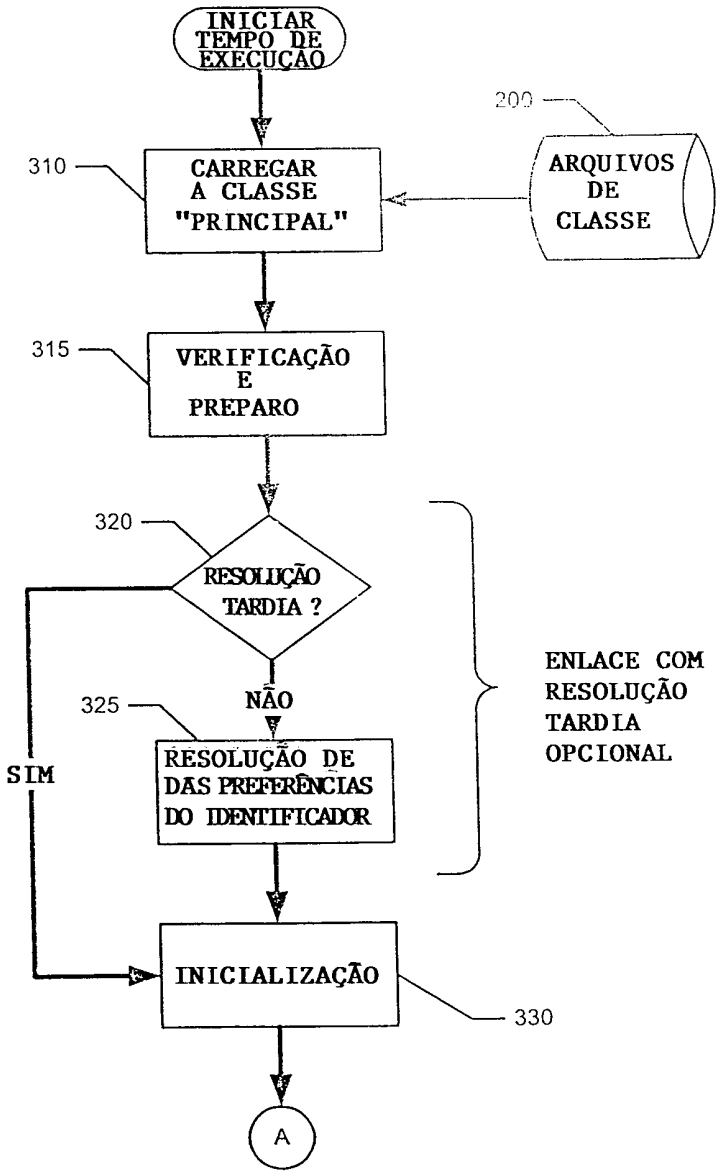


Fig. 3A
ESTADO DA TÉCNICA

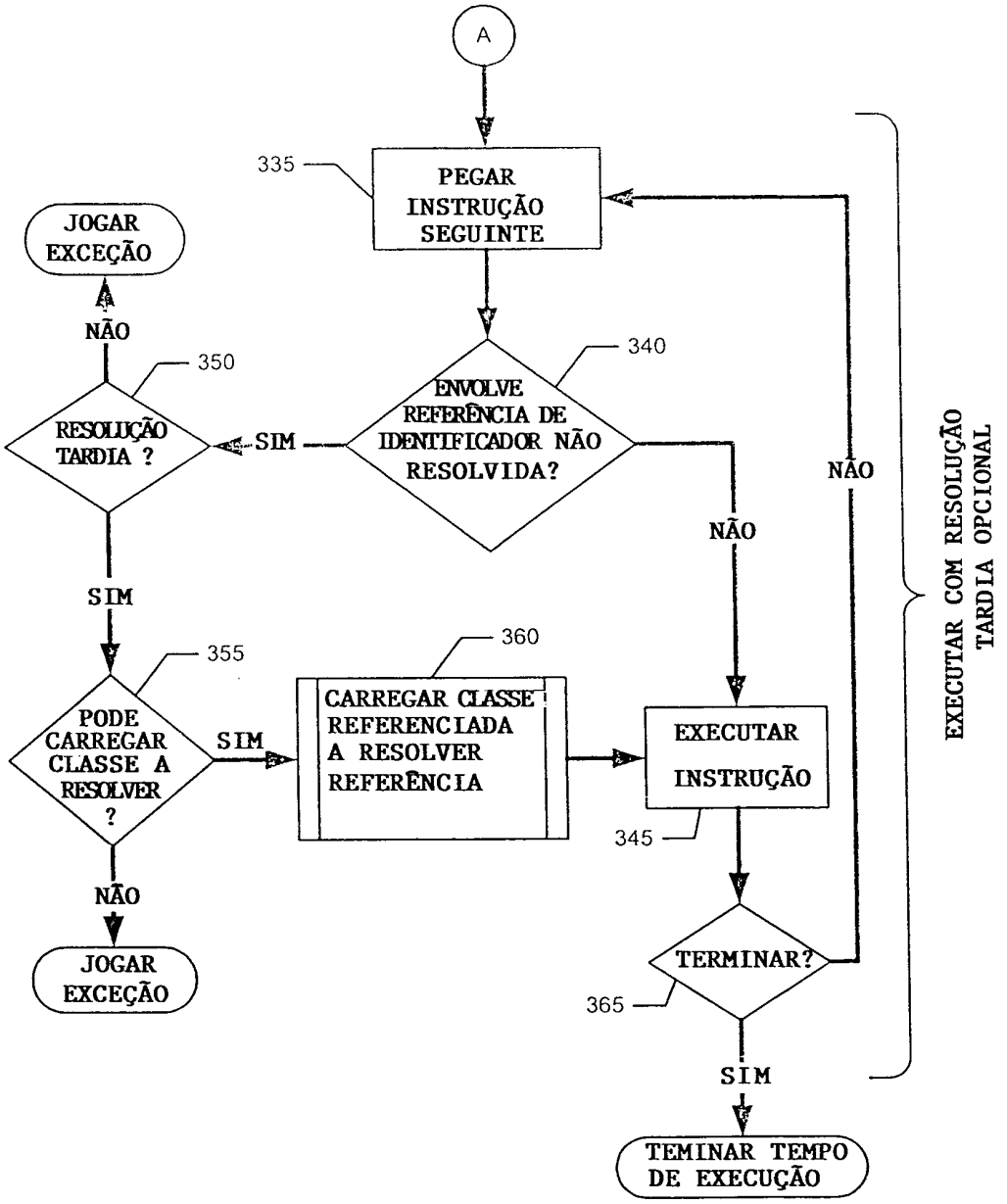


Fig. 3B
ESTADO DA TÉCNICA

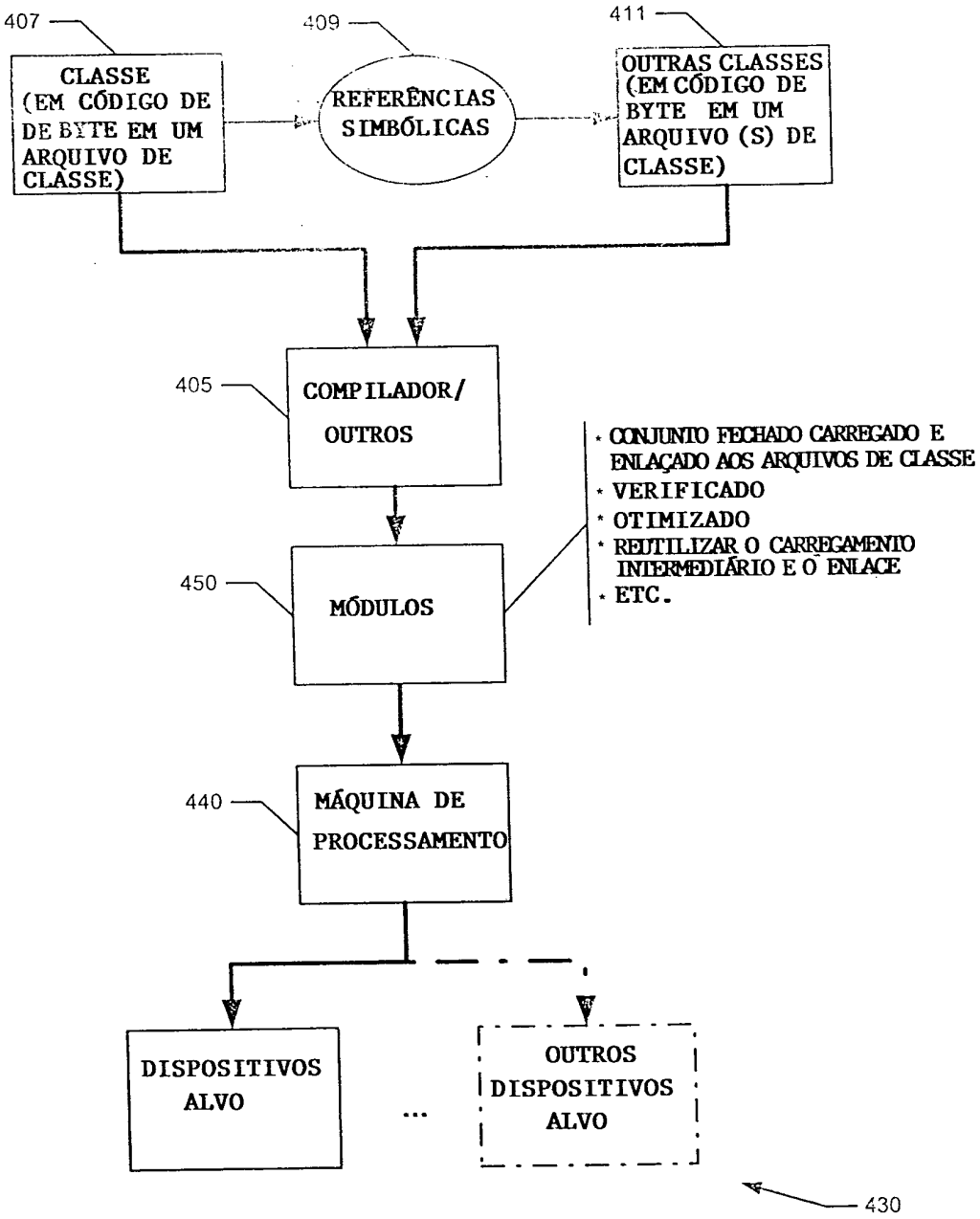
407
CLASSE (EM CÓDIGO DE DE BYTE EM UM ARQUIVO DE CLASSE)
409
REFERÊNCIAS SIMBÓLICAS
411
OUTRAS CLASSES (EM CÓDIGO DE BYTE EM UM ARQUIVO (S) DE CLASSE)
405
COMPILADOR/ OUTROS
* CONJUNTO FECHADO CARREGADO E ENLAÇADO AOS ARQUIVOS DE CLASSE
* VERIFICADO
* OTIMIZADO
* REUTILIZAR O CARREGAMENTO INTERMEDIÁRIO E O ENLACE
* ETC.
450
MÓDULOS
440
MÁQUINA DE PROCESSAMENTO
DISPOSITIVOS ALVO
...
OUTROS DISPOSITIVOS ALVO
430

Fig. 4

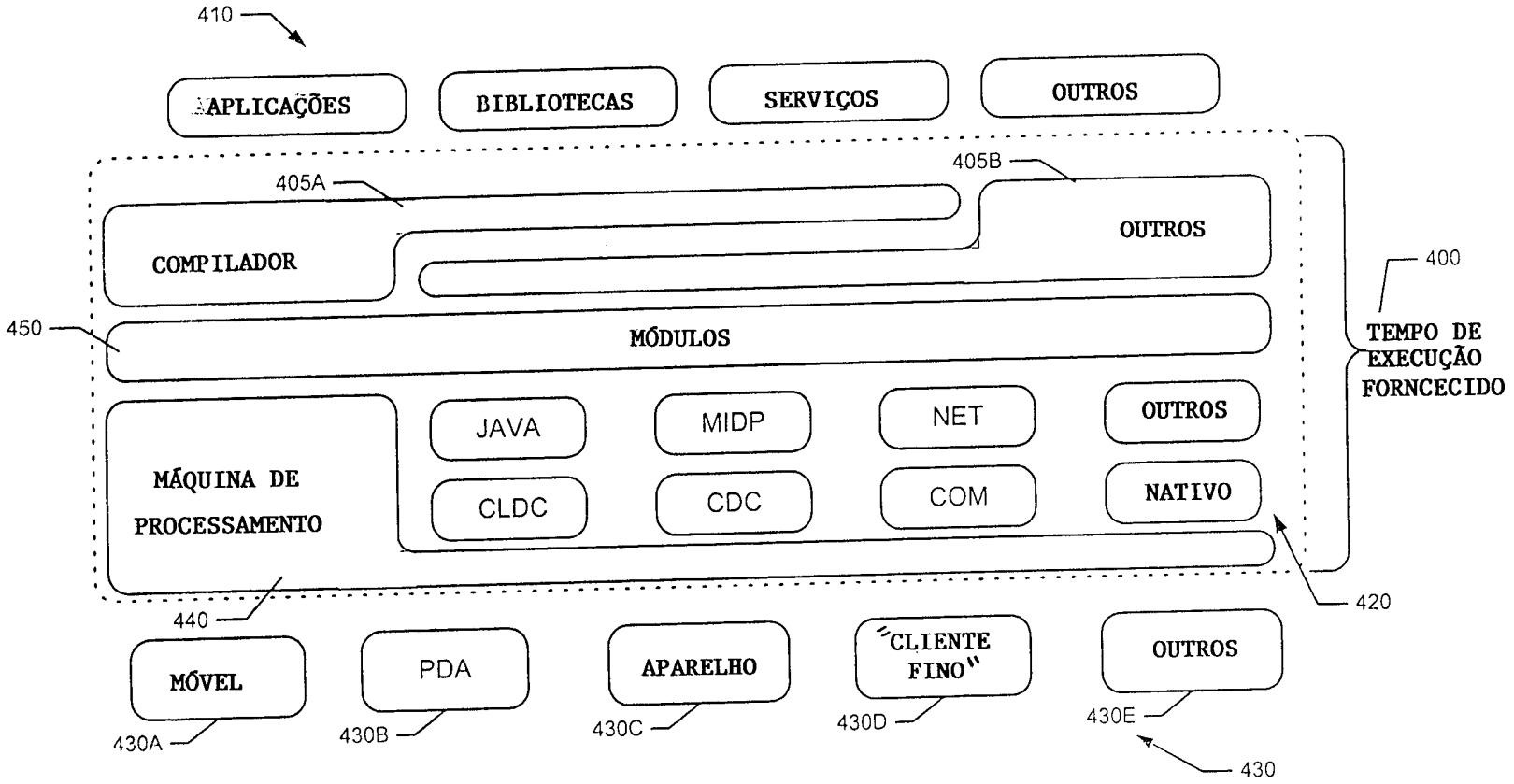
410
APLICAÇÕES
BIBLIOTECAS
SERVIÇOS
OUTROS
405A
405B
COMPILADOR
OUTROS
450
MÓDULOS
400
TEMPO DE EXECUÇÃO FORNCECIDO
MÁQUINA DE PROCESSAMENTO
JAVA
MIDP
NET
OUTROS
CLDC
CDC
COM
NATIVO
420
440
MÓVEL
PDA
APARELHO
"CLIENTE FINO"
OUTROS
430A
430B
430C
430D
430E
430

Fig. 5

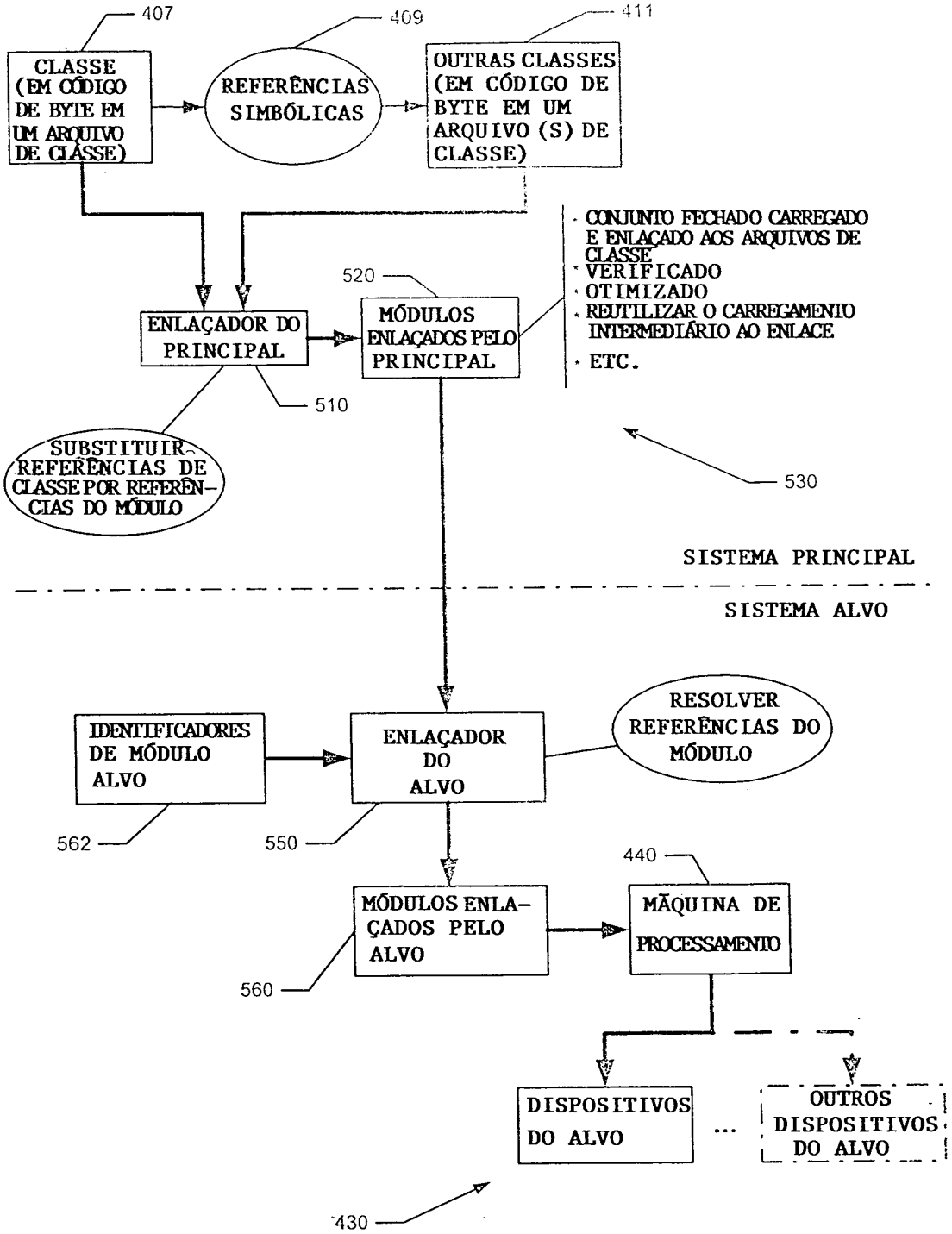
407
409
411
CLASSE (EM CÓDIGO DE BYTE EM UM ARQUIVO DE CLASSE)
REFERÊNCIAS SIMBÓLICAS
OUTRAS CLASSES (EM CÓDIGO DE BYTE EM UM ARQUIVO (S) DE CLASSE)
· CONJUNTO FECHADO CARREGADO E ENLAÇADO AOS ARQUIVOS DE CLASSE
· VERIFICADO
· OTIMIZADO
· REUTILIZAR O CARREGAMENTO INTERMEDIÁRIO AO ENLACE
· ETC.
520
ENLAÇADOR DO PRINCIPAL
MÓDULOS ENLAÇADOS PELO PRINCIPAL
510
SUBSTITUIR REFERÊNCIAS DE CLASSE POR REFERÊNCIAS DO MÓDULO
530
SISTEMA PRINCIPAL
SISTEMA ALVO
IDENTIFICADORES DE MÓDULO ALVO
ENLAÇADOR DO ALVO
RESOLVER REFERÊNCIAS DO MÓDULO
562
550
440
MÓDULOS ENLAÇADOS PELO ALVO
MÁQUINA DE PROCESSAMENTO
560
DISPOSITIVOS DO ALVO
...
OUTROS DISPOSITIVOS DO ALVO
430

Fig. 6

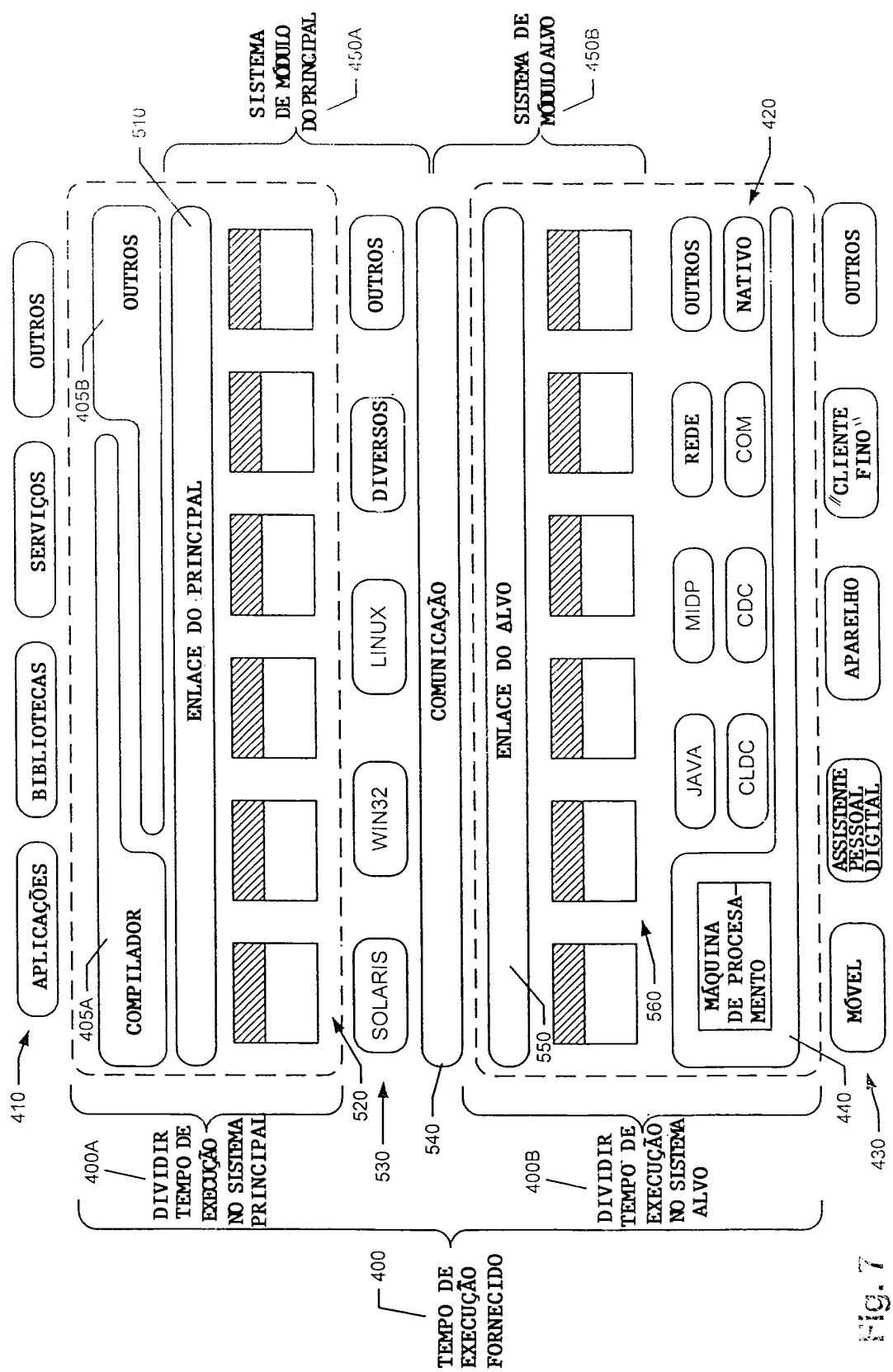
TEMPO DE EXECUÇÃO FORNECIDO
400
DIVIDIR TEMPO DE EXECUÇÃO NO SISTEMA PRINCIPAL
400A
DIVIDIR TEMPO DE EXECUÇÃO NO SISTEMA ALVO
400B
410
APLICAÇÕES
BIBLIOTECAS
SERVIÇOS
OUTROS
405A
COMPILADOR
405B
OUTROS
510
ENLACE DO PRINCIPAL
SISTEMA DE MÓDULO DO PRINCIPAL
450A
SISTEMA DE MÓDULO ALVO
450B
420
520
SOLARIS
WIN32
LINUX
OUTROS
530
COMUNICAÇÃO
DIVERSOS
540
ENLACE DO ALVO
550
560
MÁQUINA DE PROCESAMENTO
JAVA
CLDC
MIDP
CDC
REDE
COM
OUTROS
NATIVO
440
MÓVEL
ASSISTENTE PESSOAL DIGITAL
APARELHO
"CLIENTE FINO"
OUTROS
430

Fig. 7

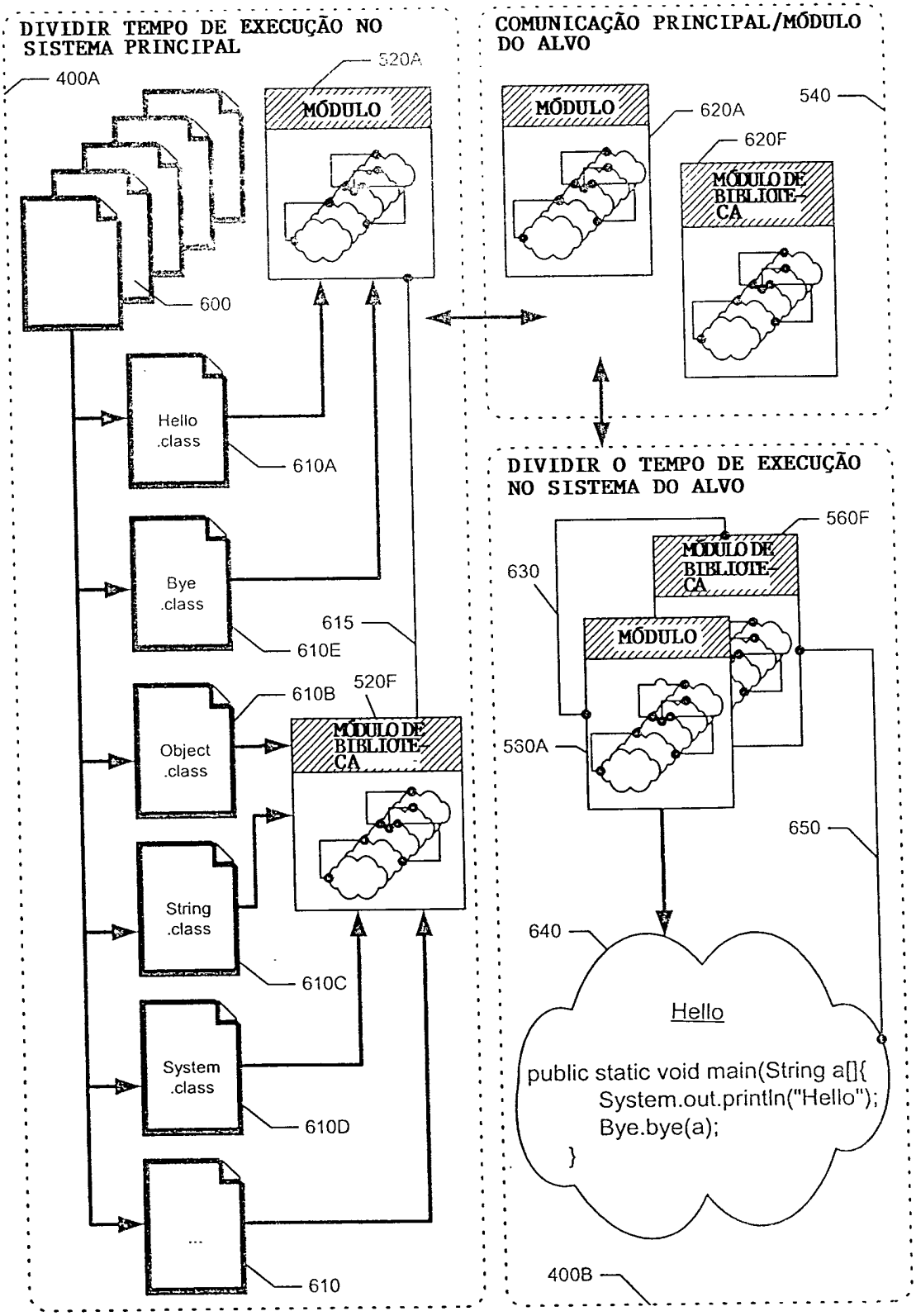
DIVIDIR TEMPO DE EXECUÇÃO NO SISTEMA PRINCIPAL
400A
520A
MÓDULO
600
Hello .class
610A
Bye .class
610E
615
610B
520F
MÓDULO DE BIBLIOTE-CA
Object .class
String .class
610C
System .class
610D
...
610
COMUNICAÇÃO PRINCIPAL/MÓDULO DO ALVO
MÓDULO
620A
540
620F
MÓDULO DE BIBLIOTE-CA
DIVIDIR O TEMPO DE EXECUÇÃO NO SISTEMA DO ALVO
560F
MÓDULO DE BIBLIOTE-CA
630
MÓDULO
560A
650
640
Hello
public static void main(String a[]{
System.out.println("Hello");
Bye.bye(a);
}
400B

Fig. 8

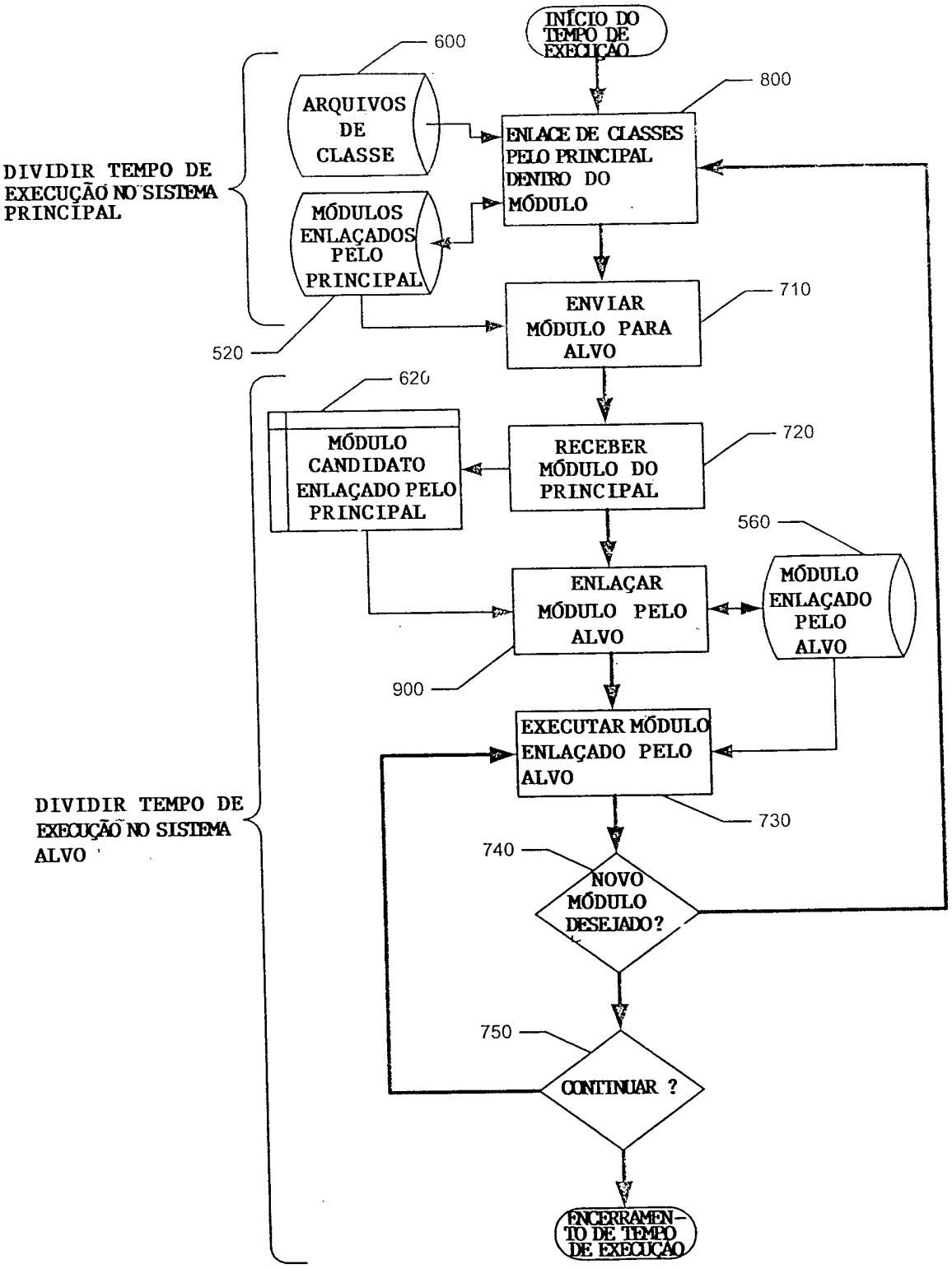
INÍCIO DO TEMPO DE EXECUÇÃO
600
ARQUIVOS DE CLASSE
800
ENLACE DE CLASSES PELO PRINCIPAL DENTRO DO MÓDULO
DIVIDIR TEMPO DE EXECUÇÃO NO SISTEMA PRINCIPAL
MÓDULOS ENLAÇADOS PELO PRINCIPAL
710
ENVIAR MÓDULO PARA ALVO
520
620
MÓDULO CANDIDATO ENLAÇADO PELO PRINCIPAL
720
RECEBER MÓDULO DO PRINCIPAL
560
ENLAÇAR MÓDULO PELO ALVO
MÓDULO ENLAÇADO PELO ALVO
900
EXECUTAR MÓDULO ENLAÇADO PELO ALVO
DIVIDIR TEMPO DE EXECUÇÃO NO SISTEMA ALVO
730
740
NOVO MÓDULO DESEJADO?
750
CONTINUAR ?
ENCERRAMENTO DE TEMPO DE EXECUÇÃO

Fig. 9

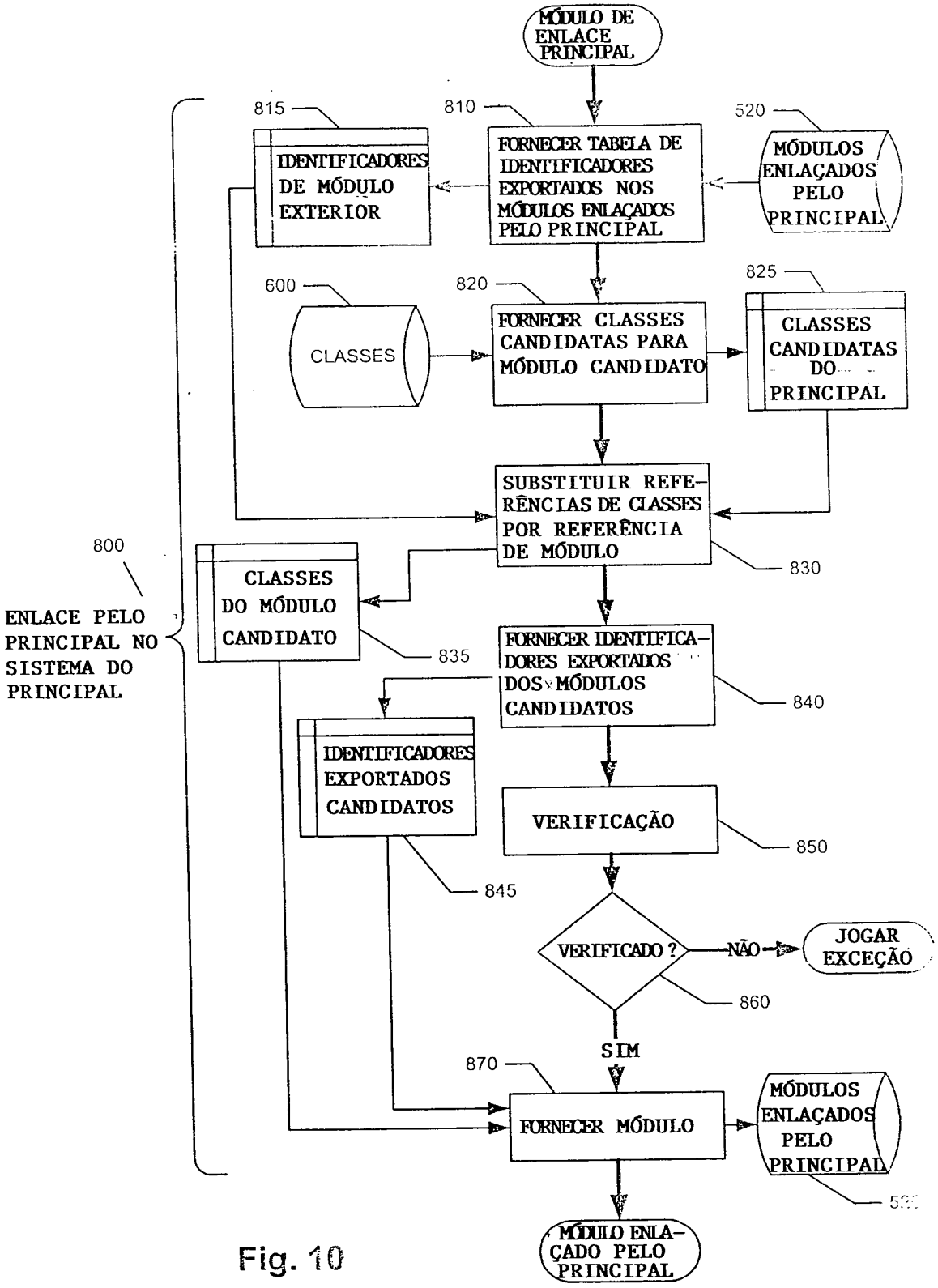
MÓDULO DE ENLACE PRINCIPAL
815
IDENTIFICADORES DE MÓDULO EXTERIOR
810
FORNECER TABELA DE IDENTIFICADORES EXPORTADOS NOS MÓDULOS ENLAÇADOS PELO PRINCIPAL
520
MÓDULOS ENLAÇADOS PELO PRINCIPAL
600
CLASSES
820
FORNECER CLASSES CANDIDATAS PARA MÓDULO CANDIDATO
825
CLASSES CANDIDATAS DO PRINCIPAL
SUBSTITUIR REFERÊNCIAS DE CLASSES POR REFERÊNCIA DE MÓDULO
830
800
ENLACE PELO PRINCIPAL NO SISTEMA DO PRINCIPAL
CLASSES DO MÓDULO CANDIDATO
835
FORNECER IDENTIFICADORES EXPORTADOS DOS MÓDULOS CANDIDATOS
840
IDENTIFICADORES EXPORTADOS CANDIDATOS
845
VERIFICAÇÃO
850
VERIFICADO ?
NÃO
JOGAR EXCEÇÃO
860
SIM
870
FORNECER MÓDULO
MÓDULOS ENLAÇADOS PELO PRINCIPAL
MÓDULO ENLAÇADO PELO PRINCIPAL

Fig. 10

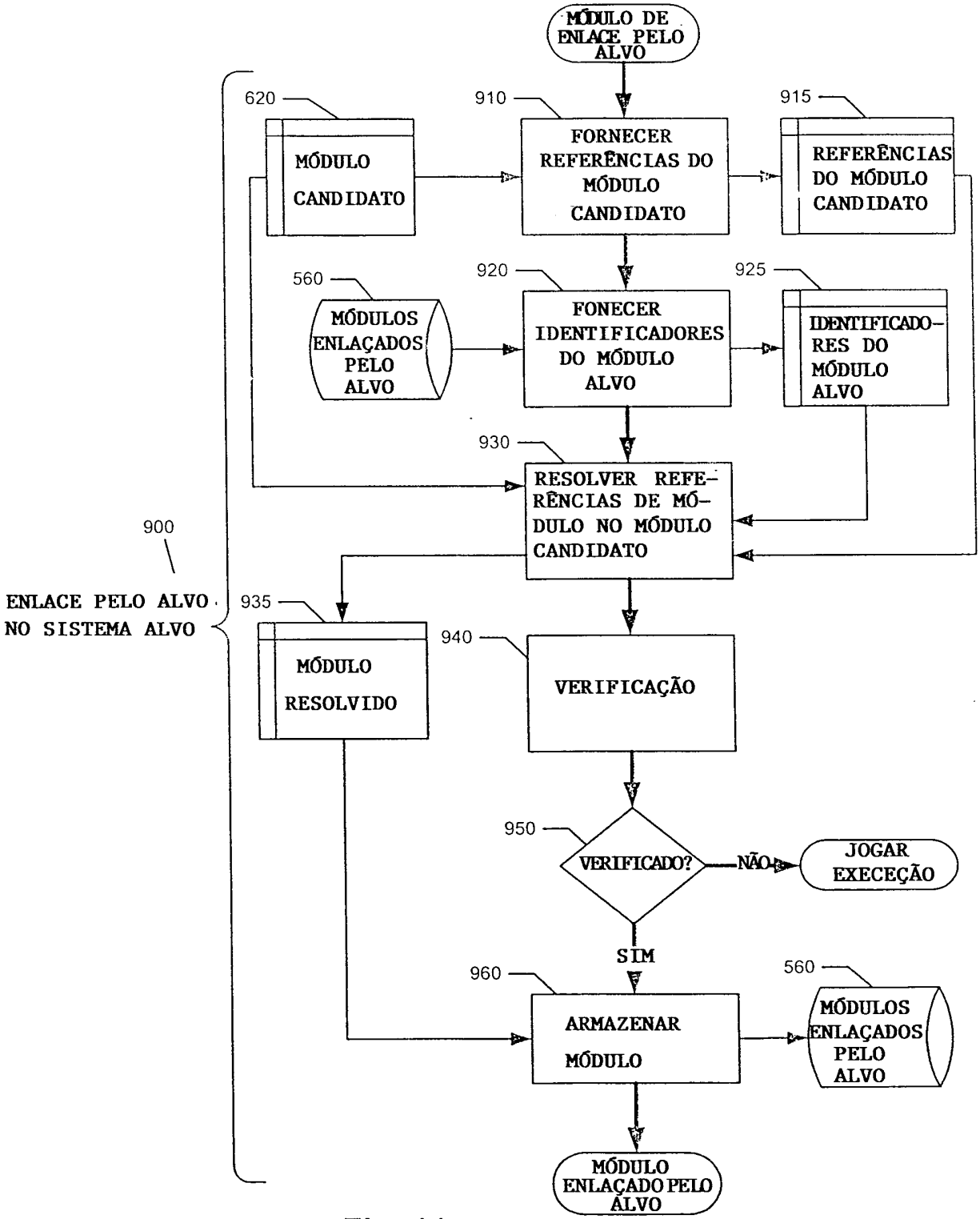
MÓDULO DE ENLACE PELO ALVO
620
MÓDULO CANDIDATO
910
FORNECER REFERÊNCIAS DO MÓDULO CANDIDATO
915
REFERÊNCIAS DO MÓDULO CANDIDATO
560
MÓDULOS ENLAÇADOS PELO ALVO
920
FONECER IDENTIFICADORES DO MÓDULO ALVO
925
IDENTIFICADORES DO MÓDULO ALVO
930
RESOLVER REFERÊNCIAS DE MÓDULO NO MÓDULO CANDIDATO
900
ENLACE PELO ALVO NO SISTEMA ALVO
935
MÓDULO RESOLVIDO
940
VERIFICAÇÃO
950
VERIFICADO?
NÃO
JOGAR EXECEÇÃO
SIM
960
ARMAZENAR MÓDULO
560
MÓDULOS ENLAÇADOS PELO ALVO
MÓDULO ENLAÇADO PELO ALVO

Fig. 11