PŘIHLÁŠKA VYNÁLEZU

zveřejněná podle § 31 zákona č. 527/1990 Sb.

(21) Číslo dokumentu:
2000 - 3437

(13) Druh dokumentu: **A3**

(51) Int. Cl.[7]:
G 06 F 9/445

(19) ČESKÁ REPUBLIKA

ÚŘAD PRŮMYSLOVÉHO VLASTNICTVÍ

(22) Přihlášeno: **12.11.1998**

(32) Datum podání prioritní přihlášky: **23.03.1998**

(31) Číslo prioritní přihlášky: **1998/98105179**

(33) Země priority: **EP**

(40) Datum zveřejnění přihlášky vynálezu: **14.11.2001**
(Věstník č. 11/2001)

(86) PCT číslo: **PCT/IB98/01799**

(87) PCT číslo zveřejnění: **WO99/49392**

(71) Přihlašovatel:
INTERNATIONAL BUSINESS MACHINES CORPORATION, Armonk, NY, US;

(72) Původce:
Baentsch Michael, Langnau, CH;
Buhler Peter, Thalwil, CH;
Oestreicher Marcus, Zurich, CH;

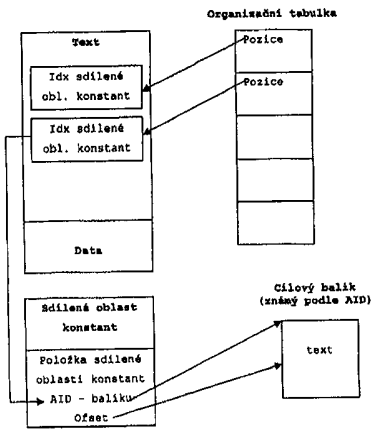
(74) Zástupce:
Kalenský Petr Dr., Hálkova 2, Praha 2, 12000;

(54) Název přihlášky vynálezu:

Java runtime systém s upravenou sdílenou oblastí konstant

(57) Anotace:

Je navržen Java runtime systém, který obsahuje interpret založený na zásobníku, provádějící program, který obsahuje bajtkódy a struktury tříd. Systém dále obsahuje upravenou sdílenou oblast konstant s interními informacemi, které se mají zachovat pro pozdní vazbu v kódu. Interní informace se po slinkování z upravené sdílené oblasti konstant odstraní.



CZ 2000 - 3437 A3

JUDr. Petr Kalenský
advokát
120 00 Praha 2, Hálkova 2

PV2000 - 3437

19.09.00

JAVA RUNTIME SYSTÉM S UPRAVENOU SDÍLENOU OBLASTÍ KONSTANT

Oblast techniky

Vynález se týká dynamického stahování kódu a linkování v prostředí Java runtime s omezenými prostředky, jako jsou například JavaCards.

Dosavadní stav techniky

V běžném systému Java jsou odkazy na struktury tříd (interní nebo externí) rozřešovány pomocí nepřímého vyhledávání názvů prostřednictvím takzvané sdílené oblasti konstant (constant pool). Takový přístup lze použít pouze v systému, který zajišťuje dostatek prostředků ve smyslu výpočetního výkonu a vnitřních prostředků.

V runtime systému s omezenými prostředky (např. JavaCard) není tento přístup výhodný. Místo toho by bylo možné použít rozřešené odkazy na struktury tříd tak, aby se odstranily nepřímé odkazy a bylo možné se vyhnout údržbě sdílené oblasti konstant.

Podstata vynálezu

Předmětem tohoto vynálezu je rozřešování odkazů (spojování, linkování).

Je předmětem tohoto vynálezu dosáhnout účinného linkování v prostředích Java runtime s omezenými prostředky.

Je předmětem tohoto vynálezu dosáhnout prostorově a za běhu účinného linkování v prostředích Java runtime s omezenými prostředky.

Přehled obrázků na výkresech

Vynález bude blíže vysvětlen prostřednictvím konkrétních příkladů provedení znázorněných na výkresech, na kterých představuje

obr. 1 linkování u formátu cap souboru podle JavaSoft.

obr. 2 linkování o ofset se zde navrhovaným formátem souboru.

obr. 3 linkování podle názvu se zde navrhovaným formátem souboru.

Příklady provedení vynálezu

Následuje obecný přehled tohoto vynálezu a budou popsány aspekty implementace. Java je programovací jazyk a prostředí vyvinuté firmou Sun Microsystems, Inc. 2550 Garcia Ave., Mountain View, CA 94043-1100, USA; termín "Java" je ochranná známka této společnosti.

Protože se tento popis zabývá také podrobnostmi implementace, předpokládá se, že odborníkům jsou známy

základní mechanismy virtuálního stroje Java Virtual machine a jeho implementace. Srozumitelný popis jazyka Java a jeho implementace JVM zveřejnili J. Goslin, B. Joy a G. Tele ve specifikaci "The Java Language Specification" a T. Lindholm a F. Yellin ve specifikaci "The Java Virtual machine Specification", obě publikace vydala firma Addison-Wesley Publishing Co., 1996. Povšimněte si však, že Java-Card Virtual machine, zde označovaná jako Java-Card VM, je jiná v mnoha ohledech, protože se používá pouze podskupina dodatkových bajtkódů.

1.0 Návrh formátu CAP souboru podle JavaSoft

Formát CAP souboru, jak je navržen od JavaSoft, dělí cap soubor na několik oddílů. Cap soubor v podstatě obsahuje textový a datový oddíl. Textový oddíl obsahuje struktury tříd, struktury metod a bajtkódové instrukce. Datový oddíl obsahuje statická pole cardletu.

Textový oddíl se odkazuje na všechny symboly (třídy, metody atd.) jejichž skutečné adresy nejsou známy před dobou linkování prostřednictvím ofsetu do sdílené oblasti konstant. Pro každý takový symbol obsahuje sdílená oblast konstant AID pro cílový balík a ofset tohoto symbolu v cílovém balíku (tj. balíku, do kterého se linkuje). Sdílená oblast konstant nabízí tyto ofsety pro všechny importované symboly (symboly definované v jiných balících) a pro všechny exportované symboly (symboly, které jsou definované a ke kterým se přistupuje uvnitř zaváděného cardletu).

Organizace textového oddílu, datového oddílu a sdílené oblasti konstant nabízí dostatek informací ke spuštění cardletu. Po zavedení cardletu by mohl linker procházet

sdílenou oblastí konstant a nahrazovat ofsety symbolů do jejich cílových balíků jejich skutečnými adresami. Linker musí pouze vyhledat konkrétní balík podle daného AID, přičíst jeho počáteční adresu k ofsetu symbolu a uložit tuto informaci zpět do sdílené oblasti konstant. Java-Card VM může pak použít tyto adresy ve sdílené oblasti konstant pro interpretaci. Sdílená oblast konstant však není v době běhu potřeba. Ve skutečnosti by vyžadovala jednu další nadbytečnou dereferenci k nalezení adresy symbolu během interpretace. Dále sdílená oblast konstant zabírá zbytečně prostor na kartě. Formát souboru cap podle JavaSoftu tudíž používá organizační tabulky pro jednotlivé oddíly kvůli odstranění požadavku na sdílenou oblast konstant po slinkování. Organizační tabulka tabulka obsahuje všechny pozice v textovém oddíle, kde je nutné provést přemístění. Linker prochází organizační tabulkou a bere ofset na této pozici do sdílené oblasti konstant. Pak rozřeší adresu tohoto symbolu a uloží ji na původní pozici v textovém oddíle. Interpret pak může přímo použít tyto adresy za běhu a tudíž lze po slinkování odstranit sdílenou oblast konstant. Jediné oddíly, které zbývají na kartě po slinkování jsou textový a datový oddíl.

Slinkování cardletů s předem vypočtenými a napevno zakódovanými ofsety do cílových balíků umožňuje kompaktní systém. Nenabízí však dostatek flexibility k umožnění různých implementací systémových tříd a specifikovaných doplňků na jiných kartách. Například cardlet, který byl převeden oproti systémovým třídám IBM, a doplněk JavaSoft pravděpodobně nepoběží na kartě se systémovým cardletem od JavaSoftu a doplňkovým cardletem od IBM. Hlavním důvodem toho je, že ofsety jednotlivých tříd, metod a ofsety polí instancí se budou lišit od implementace k implementaci.

Následující vynález se zakládá na myšlenkách současného návrhu JavaSoft a rozšiřuje je dalším odlehčujícím a flexibilním mechanismem linkování symbolů.

2.0 Nový formát CAP souborů

Formát cap souborů podle tohoto vynálezu také rozděluje cap soubor na různé oddíly. Textový oddíl obsahuje struktury tříd, struktury metod a instrukce v bajtkódech, datový oddíl opět statická pole. Cap soubor také v organizačních tabulkách udržuje nezbytné informace o přemístění. Existuje jedna organizační tabulka pro každý balík, ke které je cardlet nalinkován (tj. cílový balík).

Organizační tabulka opět obsahuje pozici v textovém nebo datovém oddíle, kde se má provést přemístění. V jednoduchém případě jsou tato místa také přemístěna o předem vypočtený ofset do důvěryhodných a dobře známých cílových balíků. V tomto případě najde linker počáteční adresu balíku s daným cílovým AID, přičte ofset do cílového balíku a uloží tuto hodnotu na původní pozici v textovém oddíle. Ofset do cílového balíku může být udržován buďto v organizační tabulce nebo v textovém oddíle na adrese přemístění, aby se organizační tabulka udržela menší. Přemístění o ofset lze vždy použít pro většinu odkazů v samotném zavedeném cardletu. Převodník může předpočítat tyto ofsety bez porušení slučitelnosti.

Z důvodu integrity systému by se odkazy na jiné vnější balíky neměly linkovat o předem vypočítané ofsety. Místo toho by se měl při linkování použít název nebo identifikátor pro odkazy na jiné balíky. Tyto názvy a s nimi spojené

hodnoty musí být uloženy pouze na kartě pro balíky, které jsou sdíleny mezi více aplety. Protože tyto tabulky dvojic název/hodnota by měly být co nejmenší, omezuje tento formát cap souboru názvy členů souboru tříd, ale stále nabízí možnost specifických implementací prodejců s jinými specifikacemi:

veřejné třídy: musí být pojmenovaané a exportovatelné.

veřejné statické metody a konstruktory: totéž

veřejná statická pole: totéž

virtuální metody:

Interpret najde virtuální metodu pomocí indexu do tabulky virtuálních metod. Tyto indexy a tabulku virtuálních metod by bylo zapotřebí rozřešit na kartě pokud chce prostředí JavaCard podporovat standardní Java pozdní vazbu v kódu virtuálních metod. Během sestavování tabulky metod se linker musí rozhodnout, zda metoda dědí jinou metodu v tabulce metod. Protože tyto metody by mohly být definovány na jiných cardletech, musí být zavedeno globální schéma názvů pro metody, aby bylo možné rozhodnout, zda dvě metody mají stejný typ. Protože volání kódu virtuální metody pozdní vazbou vyžaduje schéma globálních názvů a další prostředky na kartě pro rozřešování a pojmenování virtuálních metod, nepodporuje jej nový formát cap souboru. Ale i s tímto omezením zůstává dostatečný prostor pro specifické implementace balíků jádra a doplňků jinými prodejci. Programátor může volně přidávat private (soukromé) nebo public (veřejné) nebo statické metody a třídy. Může také přidávat nesoukromé metody instance pokud nejsou zděděné a deklarované jako final (konstanty). Převodník pak může

nahradit volání těchto metod přímými voláními.

pole instancí:

Počet a typy polí instancí ve třídě není obvykle definován specifikacemi a bude se lišit mezi implementacemi různých prodejců. Tudíž vazba ofsetů polí musí být řešena návrhem formátu CAP souboru. Použití polí co se týče linkování může být rozděleno do následujících třech kategorií:

předem vypočítané ofsety polí

Instrukce get-/put-field odkazuje na pole, jehož třída a všechny nadřazené třídy jsou definovány v obsahujícím cardletu. Ofset tohoto pole může pak být předem vypočten převodníkem a není nutné jej linkovat při zavádění. Pokud existuje souhlas, že třída objektu neobsahuje žádné deklarace polí, spadne pod tuto kategorii mnoho přístupů k polím.

Přístup k polím vnitřních tříd cardletu s jakoukoli nadřazenou třídou(třídami) vně cardletu.

Tento případ musí být podporován formátem cap souboru, aby se umožnila tvorba odvozených tříd, od tříd definovaných v samostatných balících. Převodník může vypočítat ofset pole vzhledem k velikosti instance nadřazené třídy vně cardletu a uložit tuto hodnotu do cap souboru. Při linkování je nutné přepočítat skutečný ofset pole na základě velikosti základní třídy. Stejné lze provést pro člen velikosti instance vnitřní třídy cardletu.

přístup k polím v nadřazených třídách vně cardletu nebo libovolné jiné externí třídě

Přestože jsou pole instancí zřídka deklarována jako public (veřejná), často jsou deklarována jako protected

(chráněná). Přístup k chráněným polím by měl být vždy udělen zadáním chráněných metod set-/-get-field, ale navrhovaný formát cap souboru také umožňuje přímý přístup k takto protected nebo public deklarovaným polím. Balík, který exportuje pole protected nebo public instance musí obsahovat název a ofset pro toto pole, aby se umožnilo symbolické linkování.

Balík, který exportuje třídy, metody nebo pole, obsahuje sdílenou oblast konstant, jejíž položky obsahují typ (třída, atd.), název a hodnotu pro tento symbol. Protože formát cap souborů nemusí podporovat schéma globálních názvů, mohou být snadno zadány názvy pro sdružené symboly. Symboly mohou být číslovány od 0 do n a mohou být rozdělovány spolu se specifikací aplikačního programového rozhraní (API). Implementátor takové specifikace bude stále schopen exportovat další třídy atd. výběrem nových názvů začínajících od n+1.

Obrázek 3 ukazuje symbolickou vazbu apletu oproti cílovému balíku. Položky v organizační tabulce obsahují typ přemístění (viz níže), ofset do textového oddílu, kam se přemísťuje, a název symbolu (který by kvůli efektivitě ukládání mohl být také uložen na přemísťované pozici). S navrhovaným schématem názvů může být název použit jako index do sdílené oblasti konstant cílového balíku. Položky ve sdílené oblasti konstant obsahují typ položky, název a přidruženou hodnotu. Linker přemístí položky v organizační tabulce podle typu položky:

Má se přemístit metoda:

Položka sdílené oblasti konstant cílového balíku s daným názvem musí být typu metody. Hodnota musí být ofset do

cílového balíku, kde je metoda definována. Linker vypočte adresu metody a upevní textový oddíl na daném ofsetu.

Má se přemístit třída:

stejné jako předtím, kromě toho, že položka sdílené oblasti konstant s indexem symbolu a název musí být typ třídy:

Má se přemístit statické pole:

stejné jako předtím, kromě toho, že položka sdílné oblasti konstant s indexem symbolu a názvem musí být typ statického pole, hodnota je ofset do datového oddílu balíku.

Má se přemístit relativní ofset pole instance nebo hodnota relativní velikosti instance třídy:

Položka sdílené oblasti konstant cílového balíku s daným názvem musí být typu třídy. Linker vypočte adresu cílové třídy a získá velikost instance třídy (posledně uvedená je členem struktury třídy). Linker přičte velikost instance do hodnoty dané na ofsetu textového oddílu a nahradí posledně uvedený součtem.

Absolutní přístup k poli do nedefinované třídy:

Položka sdílené oblasti konstant cílového balíku s daným názvem musí být typu pole instance. Hodnota této položky je absolutní ofset tohoto pole v instanci jeho třídy.

Pokud je zavedený cardlet sdílený balík a exportuje pole protected nebo public, přemístí linker jejich položky také do sdílené oblasti konstant. Jejich skutečné ofsety se přemístí stejným způsobem jako relativní ofsety polí instance.

Organizační tabulky mohou být odstraněny po dokončení linkování. Aplety nepotřebují sdílenou oblast konstant, pouze balíky, které mohou být sdíleny mezi více applety potřebují sdílenou oblast konstant pro další použití. Velikost sdílené oblasti konstant závisí na počtu exportovaných položek a na velikosti položky. Sdílená oblast konstant pro současné třídy systému JavaCard má momentálně kolem 160 položek. Položka sdílené oblasti konstant obsahuje pole typu, název a hodnotu, která může být uložena ve 4 bajtech (s navrhovaným schématem přidělování názvů by bylo možné ji omezit dokonce na tři bajty). Výsledkem toho je velikost sdílené oblasti konstant 720 bajtů pro všechny třídy systému.

Rozdíly oproti návrhu JavaSoft:

Sdílená oblast konstant JavaSoft v současnosti nerozlišuje mezi interně definovanými a externě definovanými položkami. To ztěžuje odstraňování položek ze sdílené oblasti konstant pokrývající interně definované položky.

Ofsety pole v instrukcích get-/put-field a pole velikosti instance ve struktuře třídy jsou široké pouze 8 bitů. To momentálně znemožňuje použít tyto hodnoty jako indexy do sdílené oblasti konstant, kde mohou být uloženy informace pro svázání ofsetu pole při linkování.

Kvůli těmto dvěma omezením potřebuje současný návrh JavaSoft rozšíření, aby se umožnilo nutné odlehčené schéma přidělování názvů - jako je zde navrhované.

3.0 Příloha

3.1 Vedlejší rozšíření

V některých prostředích nemusí existovat vůbec žádná

potřeba nebo prostor pro symbolické informace na kartě, např. by mohlo být při linkování dostatečné použít pouze předpočítané ofsety. Formát cap souboru může být v tomto směru podle tohoto vynálezu rozšířen bezpečným způsobem:

Balík nebo applet neobsahuje pouze číslo verze, ale také id prodejce (nebo id implementace).

Položky v organizačních tabulkách, které odkazují na položky ve sdílené oblasti konstant také obsahují ofsety, které by převodník mohl vypočítat v době převodu

Na začátku zavádění zavadeč ověří, zda balíky, které jsou vyžadovány appletem, jsou od stejného prodejce (např. stejná implementace) jako jsou balíky právě nainstalované na kartě. Pokud je to pravda, použije zavaděč ofsety v cap souboru appletu při linkování. Jinak zavádění selže.

3.2 Specifikace pro odkazované položky formátu cap souboru

Účel podrobného popisu není přesně specifikovat obsah cap souboru. Místo toho poskytuje podrobný popis toho, co by mělo být a co musí být v cap souboru začínající návrhem JavaSoftu. Nicméně poskytujeme také formálnější specifikace položek formátu cap souboru popsaná zde pomocí deklarací ve stylu C. Ty jsou zde specifikovány kvůli srozumitelnosti a v neoptimalizované formě:

```
// formát organizační (fixup) tabulky
typedef struct _fixup_table {
      u1 target_aid_cnt;
      u1 target_aid[];
      u1 entries_cnt;
```

```
      fixup_entry_t entries[];
};

// uspořádání organizační položky
typedef struct _fixup entry {
  u1 type;
  u2 offset;                    //ofset do textu nebo dat
  union {
    u2 target_offset;   // ofset do cílového balíku
    u2 symbol_name; // název a index do cílové sdílené oblasti konstant
  } value;
} fixup_entry_t;

// jednotlivé typy organizačních (fixup) položek
// přemísťují 16bitovou adresu v textovém nebo datovém oddíle o ofset,
// to může pokrývat adresy tříd, metod, atd.
#define RELO_TEXT_16BIT_WITH_OFFSET
#define RELO_TEXT_16BIT_WITH_OFFSET
// přemístit hodnotu symbolu do textového oddílu
#define RELO_CLASS_BY_SYM
#define RELO_METHOD_BY_SYM
#define RELO_STATICFIELD_BY_SYM
#define RELO_INSTANCEFIELD_BY_SYM

// uspořádání sdílené oblasti konstant
typedef struct _constant_pool {
      u2 cnt;
      cp_item_t items[cnt];
};

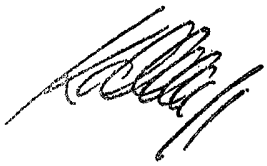
// uspořádání položky sdílené oblasti konstant
typedef struct _cp_item {
unsigned type:4          // typ položky
unsigned name:12;        // název, pokud je vůbec zapotřebí
u2 offset;               // ofset
};

// typy položek sdílené oblasti konstant
#define CONSTPOOL_CLASS
#define CONSTPOOL_METHOD
```

19.09.00

```
#define CONSTPOOL_STATICFIELD
#define CONSTPOOL_INSTANCEFIELD
```

Zastupuje:
Dr. Petr Kalenský v.r.



JUDr. Petr Kalenský
advokát
120 00 Praha 2, Hálkova 2

PV2000-3437

19.09.00

PATENTOVÉ NÁROKY

1. Java runtime systém obsahující interpret, založený na zásobníku, provádějící program obsahující bajtkódy a struktury tříd, přičemž systém ***se vyznačuje tím***, že dále obsahuje upravenou sdílenou oblast konstant s interními informacemi, které se používají pouze při linkování a s externími informacemi, které se mají zachovat pro pozdní vazbu v kódu, kde interní informace se odstraňují z upravené sdílené oblasti konstant po slinkování.

2. Java runtime systém podle nároku 1, ***vyznačující se tím***, že je JavaCard.

3. Způsob pro zavádění nového kódu do Java runtime systému, který obsahuje interpret založený na zásobníku pro provádění balíku obsahujícího bajtkódy a třídy struktur, ***vyznačující se*** kroky

slinkování (vnitřní slinkování) nového kódu do balíku existujícího v systému, čímž ofset nového kódu vzhledem k sobě je nahrazen odkazem známým pouze v době linkování,

kde externí odkazy jsou rozřešovány s použitím symbolických informací (názvů) obsažených ve sdílené oblasti konstant balíku,

odstranění alespoň informací ze sdílené oblasti konstant, která byla použita pro náhradu odkazy známými pouze v době linkování.

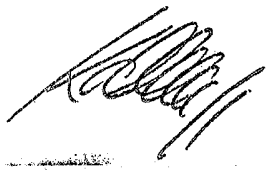
4. Způsob podle nároku 3, ***vyznačující se tím***, že jsou ze sdílené oblasti konstant odstraněny také informace použité pro linkování na externí balíky.

19.09.00

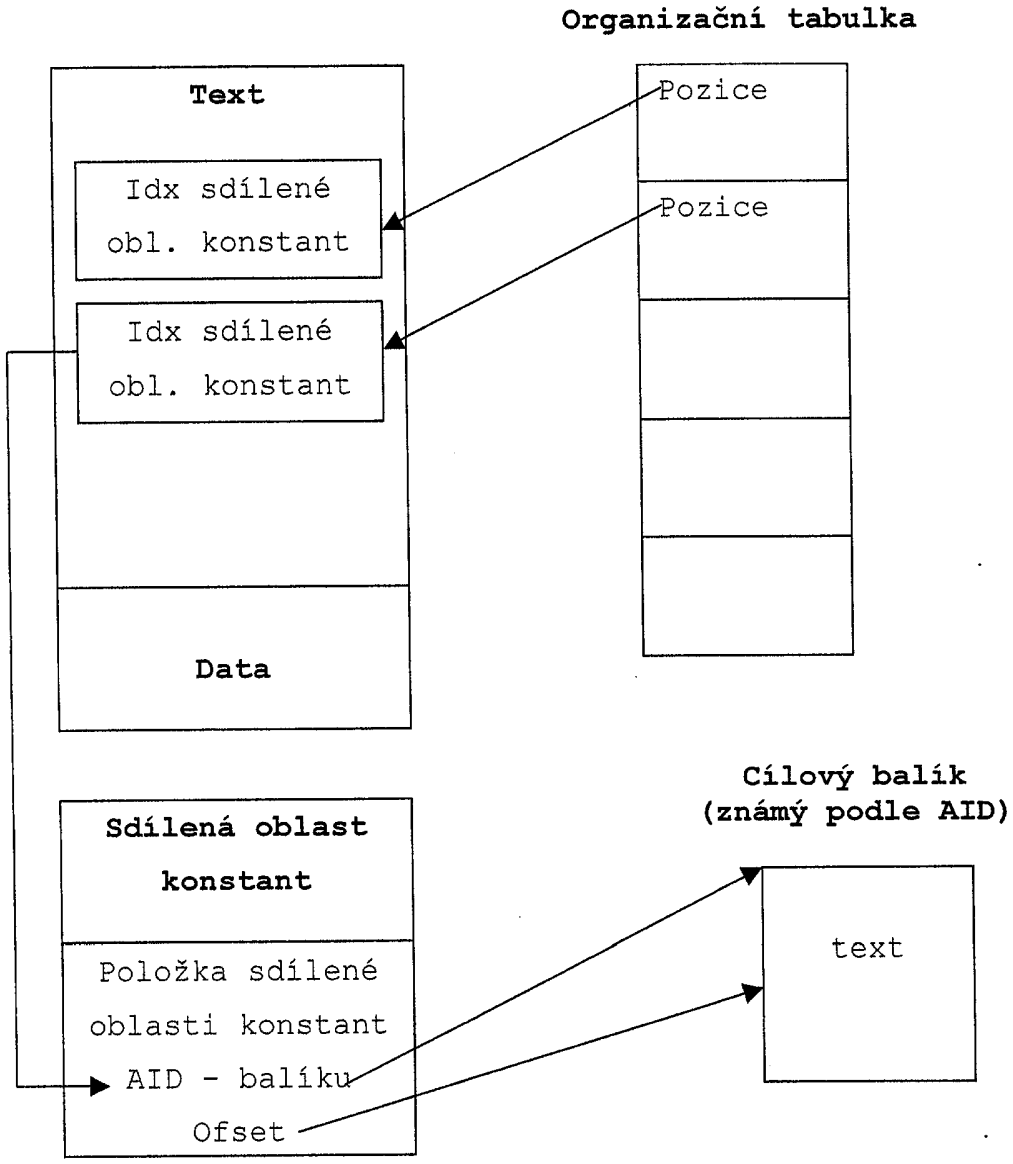
5. Způsob podle nároku 3, ***vyznačující se tím***, že zbývající informace ve sdílené oblasti konstant umožňují, aby jiné balíky a aplety, které se zavedou později,. správně přistupovaly k novému kódu (pozdní vazba v kódu).

6. Způsob podle nároku 3, ***vyznačující se tím***, že Java runtime systém je JavaCard.

Zastupuje:
Dr. Petr Kalenský v.r.



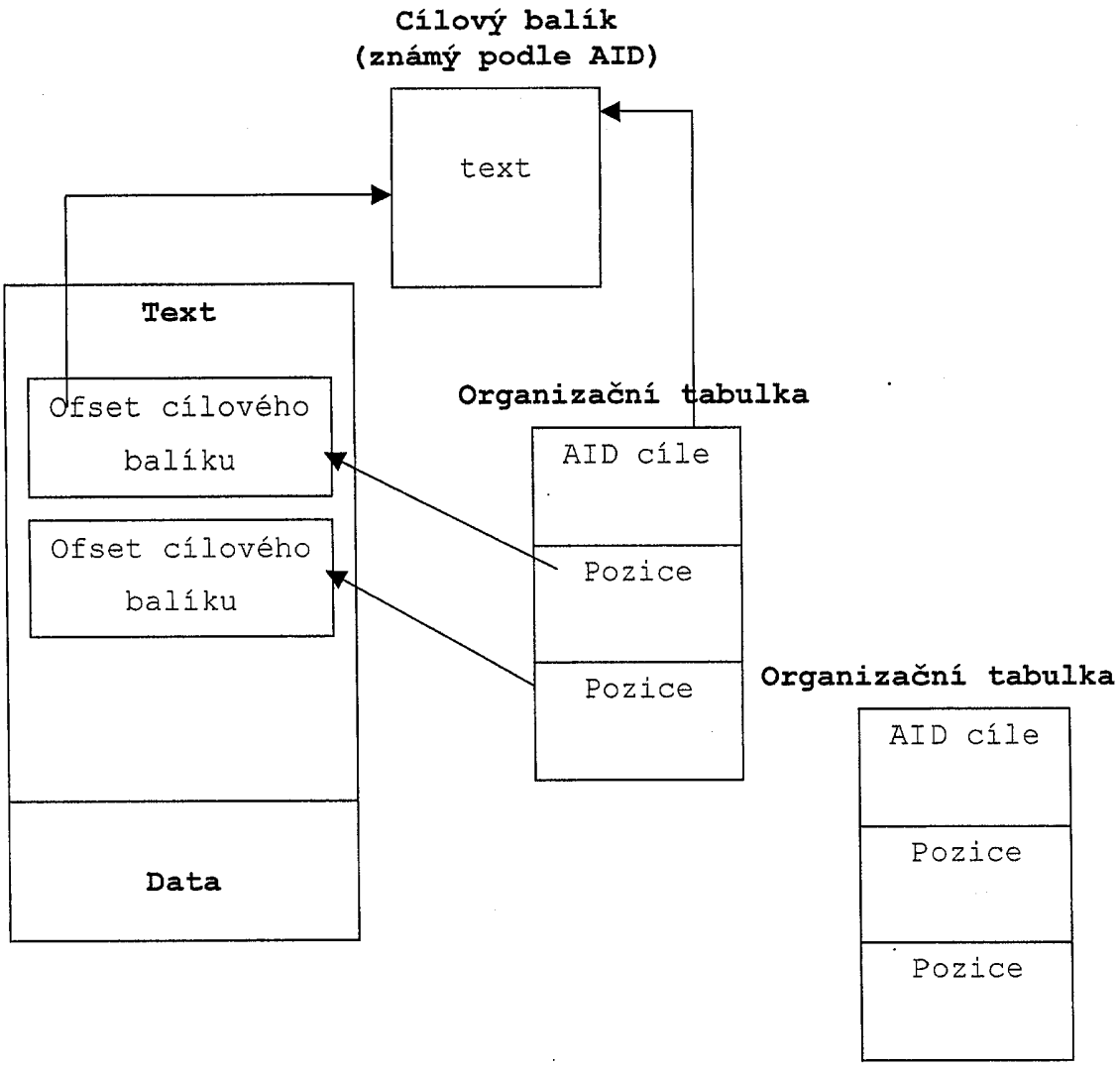
PV 2000 - 343
19.09.00



Obr. 1

PETR KALENSKÝ
ATTORNEY AT LAW

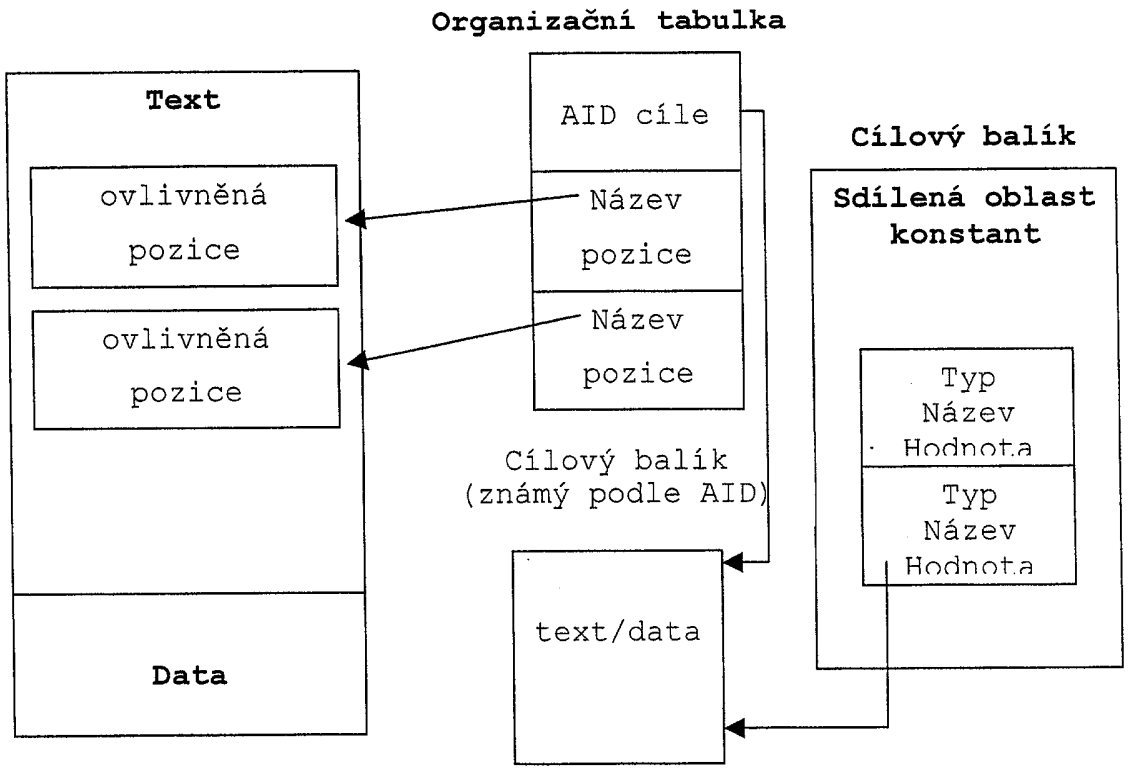
PV 2000 - 3437
19.09.00



Obr. 2

PETR KALENSKÝ
ATTORNEY AT LAW

PV 2000 - 3437
19.09.00



Obr. 3

PETR KALENSKÝ
ATTORNEY AT LAW