VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

APLIKACE V JAZYCE JAVA OVLÁDANÉ PŘES WWW

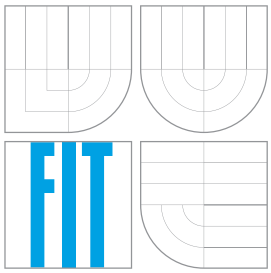
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ HOMOLA

BRNO 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

APLIKACE V JAZYCE JAVA OVLÁDANÉ PŘES WWW

JAVA APPLICATIONS CONTROLLED OVER WWW

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ HOMOLA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. RADEK BURGET, Ph.D.

BRNO 2007

Zadání diplomové práce

Řešitel	**Homola Tomáš, Bc.**
Obor	Informační systémy
Téma	**Aplikace v jazyce Java ovládaná přes WWW**
Kategorie	Web

Pokyny:

1. Seznamte se se způsobem tvorby grafického uživatelského rozhraní a appletů v jazyce Java
2. Prostudujte možnosti využití jazyka XML pro komunikaci mezi aplikacemi
3. Navrhněte způsob komunikace aplikace běžící na serveru a aplikací zajišťující GUI
4. Implementujte serverovou část komunikace jako knihovnu kompatibilní s knihovnami AWT nebo Swing
5. Implementujte klientskou část jako applet
6. Zhodnoťte možná omezení a navrhněte pokračování projektu

Literatura:

- Spell, B.: Java programujeme profesionálně. Computer Press, Praha, 2002

Při obhajobě semestrální části diplomového projektu je požadováno:

- Body 1 až 3

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese
`http://www.fit.vutbr.cz/info/szz`

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí	**Burget Radek, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání	28. února 2006
Datum odevzdání	22. května 2007

Licenční smlouva

Licenční smlouva v kompletním znění je uložena v archivu Fakulty informačních technologií Vysokého učení technického v Brně.

Abstrakt

Cílem je navržení a implemetace knihovny pro tvorbu aplikací, které bude možné ovládat vzdáleně přes WWW. Tato knihovna bude poskytovat virtualní grafické uživatelské rozhraní a bude zajišťovat komunikaci se skutečným vzdáleným grafickým uživatelským rozhraním, které bude navrženo jako applet aplikace v jazyce Java a se kterým bude možné ovládat vzdálenou aplikaci. Návrh řešení je založen na vzdáleném volání metod (RMI), které bude zajišťovat komunikaci mezi appletem a aplikací běžíci na serveru.

Klíčová slova

applet, RMI, tvorba GUI v Javě, Swing, webová aplikace

Abstract

The goal is to design and to implement a library for developing Java applications controlled over WWW. This library provides a virtual graphical user interface and it implements the network communication with a real remote graphical user interface. The remote graphical user interface is implemented as a Java applet. The proposed solution is based on the Remote Method Invocation (RMI) framework that provides the communication between the applet and the application running on the server.

Keywords

applet, RMI, GUI creation in JAVA, Swing, web application

Citace

Tomáš Homola: Aplikace v jazyce Java ovládané přes WWW, diplomová práce, Brno, FIT VUT v Brně, 2007

Aplikace v jazyce Java ovládané přes WWW

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Radka Burgeta, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Homola
22. května 2007

Obsah

Kapitola 1

Úvod

Při tvorbě softwarových aplikací s grafickým uživatelským rozhraní se v současnosti používají dva základní přístupy k realizaci tohoto rozhraní. U lokálně instalovaných aplikací se jedná desktopovou aplikaci a u aplikací typu klient – server se používá převážně webové rozhraní.

Výhoda desktopové aplikace je, že její uživatelské rozhraní, i složitější rozhraní, lze navrhnout a implementovat jednoduše. Výsledná aplikace zpravidla nemívá žádná omezení pro vykonávání svého programu na počítači, kde byla instalovaná. Naproti má nevýhody, protože má problém s přenositelností a se sdílením mezi více uživateli.

Webové aplikace s WWW rozhraním nemají problém s přenositelností ani se sdílením mezi uživateli. Ale návrh uživatelského rozhraní pro tento typ aplikací je složité a výsledné uživatelské rozhraní nenabízí takové možnosti a takovou přívětivost prostředí, jako nabízí běžné desktopové aplikace. Je to dáno tím, že HTML nenabízí takové grafické uživatelské komponenty, které mají běžné deskotopové aplikace standardně. Mezi takové komponenty patří Menu, PopupMenu, vícesloupcové seznamy, editovatelné tabulky, zobrazení stromových struktur, záložky, komponenta pro výběr datumu a další. Tyto pokročilé komponenty je si nutné, dosti složitě, pomocí prostředků HTML/Javascript/CSS implementovat a navíc nám zde kladou omezení i některé internetové prohlížeče, protože mají různou podporu pro některé vlastnosti a také chování některých vlastností. Ale je i problém s prvky uživatelské rozhraní, které HTML nabízí, a to v případě pokud chceme přes tyto prvky zobrazit v HTML vytvořené popumenu, tak se nám tyto prvky zobrazí i v tomto popupmenu.

Pokud si vybereme pozitivní vlastnosti z obou typů předchozích aplikací, jako je vzhled a jednoduchý návrh uživatelského rozhraní u běžných deskotopových aplikací a přenositelnost, jednoduché možnosti sdílení aplikace mezi více uživateli u webových aplikací, tak nám vznikne vzdáleně ovládaná aplikace přes WWW. Implementace aplikace zůstane na serveru. Uživatelské rozhraní bude ve formě appletu, který bude spuštěn v prostředí internetového prohlížeče. Navíc nám applet umožňuje vytvořit uživatelské rozhraní, které bude nezávislé na platformě ani na internetovém prohlížeči. A navíc vzhled celého uživatelského rozhraní bude stejné na jaké jsme zvyklí v daném prostředí (MS Windows, Mac OS, Linux). A implementace takového typu aplikace bude probíhat tak, jako by to byla běžná desktopová aplikace. Jenom budu mít k dispozici ekvivalentní knihovnu ke knihovně AWT/Swing, takže bude možné použít i stávající desktopovou aplikaci a pouhou změnou importované knihovny, lze vytvořit aplikaci, která bude ovládaná vzdáleně přes WWW.

Cílem práce je navrhnout a implementovat knihovnu, která umožní tvorbu vzdáleně ovládaných aplikací. Návrh je založen na aplikačním rozhraní standardní Javovské knihovny AWT/Swing (kapitola 2), komunikace mezi klientskou a serverovou částí využívá prostředky RMI (popsané v kapitole 3). Navržená architektura je popsána v kapitole 4 a

implementační detaily jsou dále rozebrány v kapitole 5. V kapitole 6 je popsána aplikace pro demonstrování funkčnosti systému vzdáleně ovládaných aplikací. V přílohách je popis komponent pro tvorbu aplikací ovládaných přes WWW a ukázka srovnání kódu aplikace napsané jako běžná deskotopová aplikace s použitím standardní javovské knihovny AWT/Swing a kódu té stejné aplikace zapsané jako vzdáleně ovládaná aplikace.

Kapitola 2

Tvorba GUI a appletů v jazyce Java

V Javě je možné vytvořit grafické uživatelské rozhraní pomocí knihovny `java.awt` nebo `javax.swing`. Přičemž třídy, které jsou v knihovně `Swing`, jsou odvozeny od tříd z knihovny `Awt`. Dále se budu věnovat komponentám z knihovny Swing.

2.1 Kontejnery

Kontejner je správce komponent, které chceme, aby se zobrazili na nějaké jiné komponentě. Nejčastěji se používají: `JFrame`, `JDialog`, `JApplet`, `JPanel`.

Přičemž každá komponenta má svoje specifické použití:

JFrame – Obvykle se používá pro vytvoření hlavního okna aplikace. Součástí je `JRootPane` (obrázek 2.1), který je tvořen `glassPane` (`Component`) a `layeredpane` (`JLayeredPane`). `layeredpane` je ještě tvořen `contentPane` (`Component`) a volitelným `menuBar` (`JMenuBar`). `layeredpane` je rodič všech komponent, které jsou zde umístěny.

JDialog – Využívá se pro tvorbu modálních nebo nemodálních dialogových oken. Součástí je `JRootPane`.

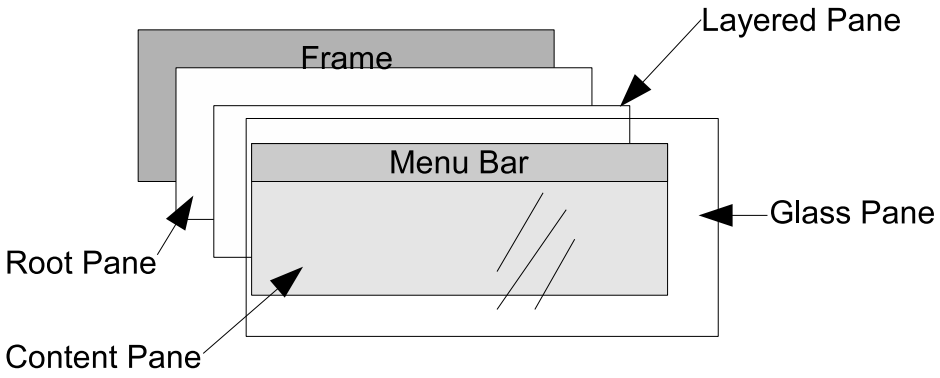
JApplet – Je vhodný pro vložení do jiné aplikace (př. webová stránka). Nemůže být použit jako samostatný program. Součástí je `JRootPane`. Více o appletech je nápsáno dále.

JPanel – Je vhodný pro umisťování souvisejících komponent na `JFrame`, `JDialog` či na jiný `JPanel`.

2.2 Správci rozvržení

Pokud si chceme zjednodušit návrh a rozmisťování komponent, které jsou vloženy do kontejneru, tak můžeme využít některého ze správců rozvržení, který se bude starat o umístění a velikost komponenty. Tím je možné vytvářet flexibilní uživatelské rozhraní, jinak si takové chování musíme sami pracně naprogramovat.

Java poskytuje několik správců rozvržení, každý z nich má jiné chování a jinak se i používá. Pokud nechceme využít služeb od žádného ze správců rozvržení, můžeme nastavit



Obrázek 2.1: Vrstvy JRootPane

jako správce rozvržení `null`, potom jednotlivé komponenty jsou umístěny staticky podle zadaných parametrů (velikost, pozice). Také máme možnost si další správce rozvržení doprogramovat, aby vyhovovali podle našich představ.

Dále představím několik základních správců rozvržení: CardLayout, FlowLayout, GridLayout, BorderLayout, GridBagLayout, BoxLayout.

2.2.1 CardLayout

Jedná se o nejjednoduší správce rozvržení. Umožňuje zobrazit v jednom okamžiku pouze jednu komponentu. Velikost komponenty je určena podle dostupné zobrazovací plochy kontejneru. Která komponenta bude zobrazena, můžeme určit pomocí metod třídy `CardLayout` `next()`, `previous()`, `first()`, `last()` a `show()`. Metoda `first()` zobrazí komponentu, která je první v seznamu. Metoda `last()` zobrazí komponentu, která je poslední v seznamu. Metody `next()`, `previous()` umožňují průchod seznamu dopředu nebo dozadu. Metoda `show()` zobrazí konkretní komponentu. Každá komponenta musí mít přiřazený jedinečný název.

2.2.2 FlowLayout

Tento správce rozvržení uspořádává komponenty podle toho v jakém jsou pořadí vkládány do kontejneru. Uspořádává je postupně zleva doprava a shora dolů. Velikosti jednotlivých komponent nemění, ponechává je tak, jak byly nastaveny. Kolik komponent umístí na jediném řádku, záleží podle toho, jaká je šířka řádku. Umožňuje nastavit, jak se mají řádky s komponentami zarovnávat pomocí konstant třídy `FlowLayout` `LEFT`, `CENTER`, `RIGHT`.

2.2.3 GridLayout

Tento správce rozvržení rozdělí prostor na mřížku rovnoměrných buněk. Do každé buňky lze umístit jednu komponentu. Jak má být prostor rozdělen, lze určit parametry třídy `GridLayout` a to počtem řádků a sloupců. Velikost každé komponenty je upravena tak, aby se vešla do buňky.

2.2.4 BorderLayout

Tento správce rozvržení rozdělí plochu na pět oblastí, kam lze vkládat komponenty (sever, jih, východ, západ a střed). Kam bude jednotlivá komponenta umístěna, je určeno konstantou třídy `BorderLayout` `NORTH`, `SOUTH`, `EAST`, `WEST`, `CENTER`. Do každé oblasti lze vložit pouze jednu komponentu. Velikost komponenty bude upravena podle velikosti jednotlivé oblasti.

2.2.5 GridBagLayout

Tento správce rozvržení je nejflexibilnější pro návrh uživatelského rozhraní, také je nejsložitější na použití. Podle názvu, tak jako `GridLayout`, rozděluje plochu na mřížku buněk. Ale zde není vyžadováno, aby každá komponenta měla stejnou velikost, čímž se velikost každého řádku a sloupce může lišit. Také podporuje nastavování omezení `GridBagConstraints`, které určí, do které buňky se komponenta vloží, a jak se bude komponenta v rámci buňky chovat. Možné je třeba zarovnaní, velikost, případně jestli bude i jedna komponenta zobrazena i přes sousední buňky. Velikost komponent je většinou určena upřednostňovanými rozměry, pokud je prostor dostatečně velký, jinak nastaví rozměry, které budou vyhovovat. Ale nikdy nestaví velikost menší, než jaká je minimální velikost komponenty.

2.2.6 BoxLayout

Tento správce rozvržení je zjednodušenou verzí `GridBagLayout`. Umožňuje zobrazovat komponenty pouze na jediném řádku nebo sloupci. Oproti ostatním správcům rozvržení, správce rozvržení `BoxLayout` využívá více i určitých vlastností komponent, jako je minimální a maximální velikost komponenty. Dále využívá odsazení na ose X a Y. Velikost komponent nastavuje podle dostupné velikosti zobrazované plochy, ale také přihlíží k minimální a maximální velikosti komponent, takže nenastaví velikost menší, než jaká je udána minimální velikostí. A naopak nenastaví velikost větší, než jakou udává maximální velikost.

2.3 Grafické komponenty

Pro tvorbu grafického uživatelského rozhraní máme k dispozici celou řadu prvků grafického uživatelského rozhraní. Různé typy tlačítek (`JCheckbox`, `JToggleButton`, `JRadioButton`, `JButton` ...), editační pole (`JTextField`, `JTextArea` ...), statické objekty (`JLabel` ...), prvky pro tvorbu menu a popupmenu (`JMenu`, `JMenuItem`, `JPopupMenu` ...), složitější prvky pro zobrazení různých seznamů (`JComboBox`, `JList`, `JTable`, `JTree` ...) a spoustu dalších prvků.

Prvky umísťujeme na nějakou zobrazovanou plochu (JFrame, JPanel ...). Nastavujeme jim, jaké události budeme zpracovávat.

2.4 Události

Pro zpracování událostí je možné použít dva modely: model pozorovatele (nebo také vydavatel/odběratel) a model dotazování. Model dotazování není příliš efektivní, protože je nutné se opakovaně dotazovat, zdali už nějaká událost nastala a jestli se tedy má provést její obsluha. Model pozorovatele je mnohem lepší. Pokud nějaká událost nastane, tak je možné okamžitě provést zpracování události. V programovacích jazycích se model pozorovatele

nazývá událostmi řízené programování. V jazyce Java je tento model také použit. Tudíž jen programujeme, co se bude dít, když nějaká událost nastane.

Není potřeba reagovat na všechny události, které nastanou. Máme možnost si vybrat na jakou událost budeme reagovat. Stačí si jen vytvořit posluchače (listener, objekt implementující rozhraní `EventListener`) a přiřadit ho komponentě, u které chceme reagovat na danou událost. V Javě existují rozšíření k obecnému rozhraní `EventListener`, dělí se podle toho, na jaké typy události se specializují. Pro události generované myší se použije rozhraní `MouseListener`, pro událostí generované nějakou akcí se použije `ActionListener`, pro události generované stisknutím nějaké klávesy se použije `KeyListener` a mnohé další.

Pokud implementujeme nějaké požadované rozhraní posluchače, pak už stačí předat tento posluchač komponentě použitím metody `addXXXListener()`, kde XXX značí název implementovaného rozhraní (pro `MouseListener` to bude `addMouseListener()`). Protože každé rozhraní posluchače nabízí mnoho metod (každá je na obsluhu jiné událost), které je nutné implementovat, i když chceme reagovat pouze jen na jednu událost, máme k dispozici adaptéry (př. `MouseAdapter`), které implementují rozhraní posluchače, tak že na událost nijak nereagují. Tím je dán prostor k odvození a překrytí jen metody, která nás zajímá.

Každé metodě pro zpracování události je samozřejmě předána informace o vzniklé události a to která komponenta ji generuje. Bázová třída pro všechny typy události je `java.util.EventObject`, ostatní třídy událostí jsou od ní odvozeny, podle toho o jaký typ události se jedná. Například pro události generované stisknutím klávesnice je předán objekt třídy `KeyEvent`, kde se dozvíme, které klávesa byla stisknuta a zdali nebylo zároveň stisknuta kombinace kláves (př. CTRL+S).

2.5 Applet

Applet je speciální kontejner pro prvky grafického uživatelského rozhraní, který může být vložen jako součást webové stránky, protože poskytuje rozhraní pro komunikaci mezi appletem a prostředím, ve kterém je spuštěn. Protože applet si je možné stáhnout jako součást nedůvěryhodných webových stránek, a pak je kód appletu spuštěn na klientském počítači, je nutné aby měl definované omezení, co může provádět a co ne. Proto je applet spuštěn v uzavřeném prostředí "pískoviště" (sandbox), které mu odmítne vykonávat nějakou podezřelou činnost. Toto prostředí zajišťuje virtuální stroj jazyka Java.

Výhody appletu je rozšíření statického obsahu webových stránek. Umožňuje tvořit složitější dynamický obsah oproti Javascriptu a navíc kód appletu je už zkompilován do byte-kódu, takže je mnohem rychlejší na interpretaci než Javascript.

Bezpečnostní politika pro applety se liší podle toho, jestli se jedná o důvěryhodný nebo nedůvěryhodný kód. Za důvěryhodný applet se považuje ten, který je digitálně podepsán, a potom má větší "pískoviště" na hraní. Může třeba přistupovat k lokálním souborům.

Seznam omezení pro nedůvěryhodné applety:

- Nelze číst soubory na klientském souborovém systému.
- Nelze zapisovat do souborů na klientském souborovém systému.
- Nelze mazat soubory na klientském souborovém systému a to buď metodou `File.delete()` nebo voláním systémového příkazu `rm` nebo `del`.
- Nelze přejmenovávat soubory na klientském souborovém systému a to buď metodou `File.renameTo()` nebo voláním systémového příkazu `mv` nebo `rename`.

- Nelze vytvářet adresáře na klientském souborovém systému a to buď metodou `File.mkdirs()` nebo voláním systémového příkazu `mkdir`.
- Nelze prohlížet obsah adresářů.
- Nelze testovat, zdali nějaký soubor existuje či ne.
- Nelze získávat informace o nějakém souboru jako je velikost, typ, datum a čas poslední modifikace.
- Nelze vytvářet síťové spojení k jakémukoli jinému počítači, kromě toho odkud byl applet stažen.
- Nelze poslouchat a přijímat síťové spojení na libovolném portu na klientském systému.
- Nelze vytvořit okno na nejvyšší úrovni bez zobrazeného upozornění, že okno bylo vytvořeno nedůvěryhodným appletem.
- Nelze jakýmkoli způsobem získávat jméno uživatele nebo jméno jeho domovského adresáře přes systémové proměnné `user.name`, `user.home`, `user.dir`, `java.home` a `java.class.path`.
- Nelze definovat jakoukoli systémovou proměnou.
- Nelze spustit jakýkoli program použitím metody `Runtime.exec()`.
- Nelze ukončit interpret Javy použitím `System.exit()` nebo `Runtime.exit()`.
- Nelze načítat dynamické knihovny na klientském systému použitím metod `load()` nebo `loadLibrary()` z tříd `Runtime` nebo `System`.
- Nelze vytvářet nebo jinak manipulovat s vlákny, které nejsou ve stejné skupině vláken (`ThreadGroup`) jako applet.
- Nelze vytvářet `ClassLoader`.
- Nelze vytvářet `SecurityManager`.
- Nelze specifikovat funkce pro síťovou kontrolu pomocí `ContentHandlerFactory`, `SocketImplFactory` nebo `URLStreamHandlerFactory`.
- Nelze definovat třídy, které jsou součástí balíků na klientském systému.

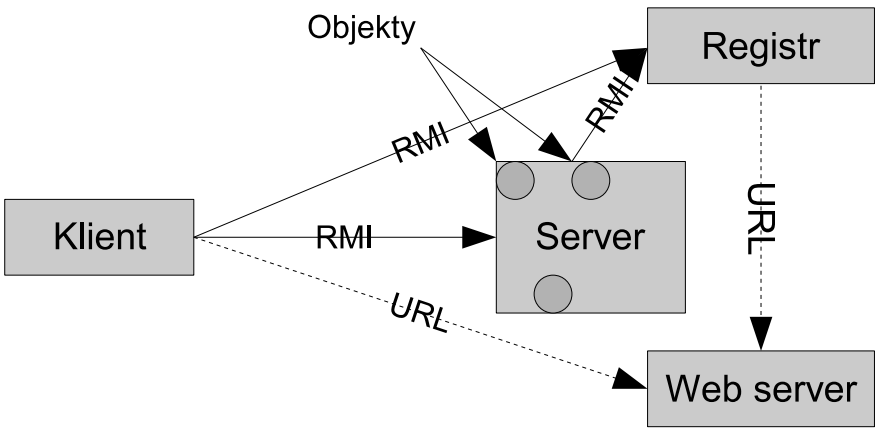
Kapitola 3

Remote Method Invokation (RMI)

Jedná se o distribuovanou technologii, která umožňuje vzdálené volání metod objektů. Podobný systém je vzdálené volání procedur (RPC).

3.1 Architektura

Distribuovaný systém je tvořen registrem, který uchovává odkazy na vzdálené objekty. Server aplikace, která nabízí objekty pro vzdálený přístup, musí odkaz na tyto objekty, pro vzdálené volání, zaregistrovat v registru. Klient, který chce získat odkaz na vzdálený objekt, se nejdříve dotáže registru, zdali požadovaný objekt je registrován, a pokud je registrován, tak mu vrátí odkaz na požadovaný objekt. Komunikace mezi jednotlivými částmi je zajištěn pomocí RMI protokolu. Na obrázku 3.1 je ještě web server, který zde vystupuje jako způsob distribuce kódu pro klienta pomocí URL protokolu (HTTP, FTP, ...).



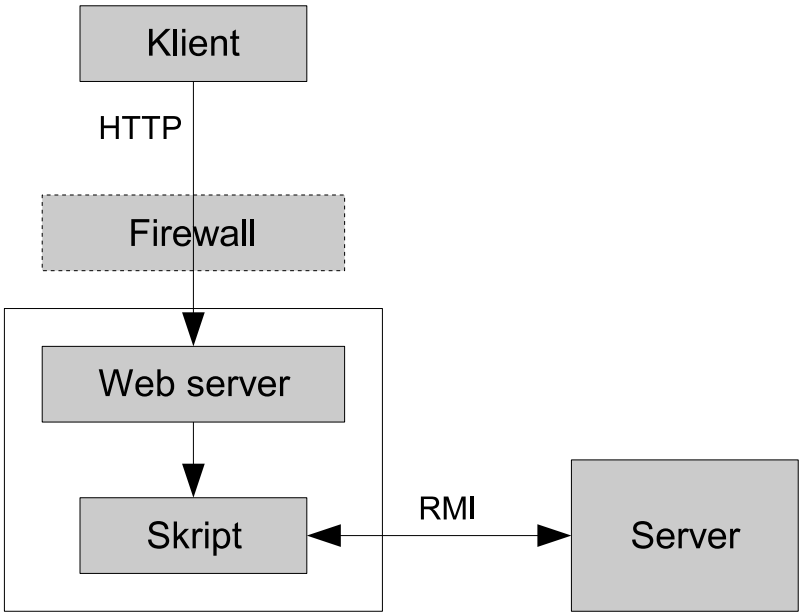
Obrázek 3.1: RMI architektura

3.2 Architektura s firewall

Standardně se RMI snaží navázat přímé spojení přes internet ke vzdálenému počítači, jenže takové pokusy mohou blokovat firewally. Ale RMI poskytuje ještě dva alternativní způsoby

spojení pomocí HTTP mechanizmu, což umožní klientovi za firewallem volat vzdálené metody.

RMI volání je posláno v těle HTTP POST požadavku, a potom výsledek je vrácen v HTTP odpovědi. Požadavek je možné poslat dvěma způsoby, záleží co umožní firewall. První možnost je, že HTTP dotaz bude poslán hostiteli na libovolný port, na kterém RMI server poslouchá. Nebo firewall umožňuje poslat HTTP požadavky jen na známé HTTP porty, pak je HTTP dotaz poslán hostiteli na port 80, kde poslouchá HTTP server a zde bude spuštěn skript, který tento požadavek zpracuje tím, že se spojí s RMI serverem a předá RMI volání ke zpracování. Po zpracovaní RMI serverem je výsledek předán zpět skriptu a ten vytvoří HTTP odpověď.



Obrázek 3.2: RMI architektura přes firewall

Vzdálené volání metod objektu přes HTTP požadavky je pomalejší než, kdyby byly posílány přes přímé spojení.

3.3 RMI protokol

Přenos zpráv se provádí pomocí dvou proudů (výstupní a vstupní). Význam proudů je ve vztahu ke klientovi. Výstupní proud se používá pro odchozí zprávy a vstupní proud pro příjem příchozích zpráv.

3.3.1 Výstupní proud

Proud je tvořen hlavičkou a pak následuje seznam zpráv. Specifikace formátu výstupního proudu je přebrána z dokumentu [1, Java^{TM} Remote Method Invocation Specification].

Out:

    Header Messages
    HttpMessage
Header:
    0x4a 0x52 0x4d 0x49 Version Protocol
Version:
    0x00 0x01
Protocol:
    StreamProtocol
    SingleOpProtocol
    MultiplexProtocol
StreamProtocol:
    0x4b
SingleOpProtocol:
    0x4c
MultiplexProtocol:
    0x4d
Messages:
    Message
    Messages Message

Jsou definovány tři typy odchozích zpráv:

Call – volání metody

Ping – testování zdali vzdálený počítač je připraven

DgcAck – potvrzení, že klient přijal vzdálené objekty v návratové hodnotě

Message:
    Call
    Ping
    DgcAck
Call:
    0x50 CallData
Ping:
    0x52
DgcAck:
    0x54 UniqueIdentifier
CallData:
    ObjectIdentifier Operation Hash Argumentsopt
ObjectIdentifier:
    ObjectNumber UniqueIdentifier
UniqueIdentifier:
    Number Time Count
Arguments:
    Value
    Arguments Value
Value:
    Object
    Primitive

HttpMessage:
    HttpPostHeader Header Message
HttpReturn:
    HttpResponseHeader Return

3.3.2 Vstupní proud

Specifikace formátu zpráv pro vstupní proud je přebrána z dokumentu [1, Java^{TM} Remote Method Invocation Specification].

Jsou definovány tři typy příchozích zpráv:

ReturnData – výsledek volání metody

HttpReturn – výsledek v HTTP protokolu

PingAck – odpověď na Ping

In:
    ProtocolAck Returns
    ProtocolNotSupported
    HttpReturn
ProtocolAck:
    0x4e
ProtocolNotSupported:
    0x4f
Returns:
    Return
    Returns Return
Return:
    ReturnData
    PingAck
ReturnData:
    0x51 ReturnValue
PingAck:
    0x53
ReturnValue:
    0x01 UniqueIdentifier Valueopt
    0x02 UniqueIdentifier Exception

3.3.3 Typy hodnot použitých v RMI protokolu

Použitý typ	Typ v Javě
Count	`short`
Exception	`java.lang.Exception`
Hash	`long`
Hostname	`UTF`
Number	`int`
Object	`java.lang.Object`
ObjectNumber	`long`
Operation	`int`
PortNumber	`int`
Primitive	`byte, int, short, long...`
Time	`long`

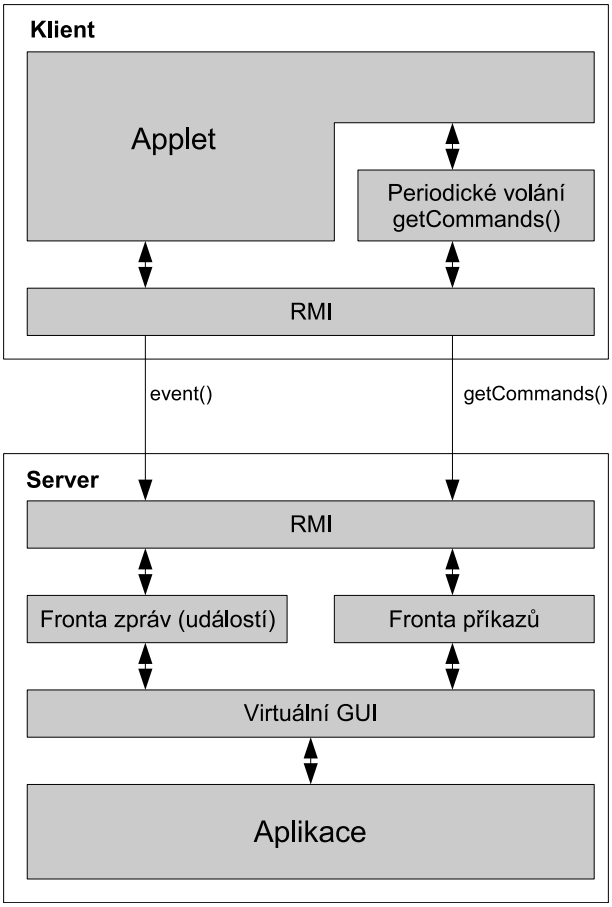
3.4 Přenos parametrů

Aby bylo možné přenášet parametry pro volání vzdálené metody nebo návratové hodnoty, musí tyto parametry implementovat rozhraní `java.io.Serializable`. Parametry, které nejsou poskytovány pro vzdálený přístup, jsou předány jako kopie. Tedy nejsou předány referencí, jak to bývá obvyklé v Javě. Proto před voláním vzdálené metody jsou pro tyto parametry a návratovou hodnoty vytvořeny kopie, které jsou pak serializované.

Kapitola 4

Návrh architektury pro vzdálené ovládání aplikací

Architektura bude navržena jako klient-server architektura, kde klienta bude představovat applet spuštěný v prostředí internetového prohlížeče a server bude aplikace s virtuálním grafickým uživatelským rozhraním. Navržená architektura je na obrázku 4.1.



Obrázek 4.1: Architektura

Klient bude tvořen z:

- appletu
- periodického dotazování
- RMI

Server bude tvořen z:

- vlastní aplikace
- virtuálního grafického uživatelské rozhraní
- fronty příkazů
- fronty zpráv
- RMI

4.1 Komunikace klient-server

Komunikaci mezi klientem a serverem bude zajišťovat RMI (Remote Method Invocation), které zajistí přenos zpráv právě voláním vzdálené metody. Komunikační rozhraní přes RMI, které bude poskytovat server, umožní získat seznam příkazů a signalizovat vzniklou událost na klientovi.

Rozhraní poskytované serverem:

```
interface ServerAppInterface extends Remote
{
    List<Command> getCommands() throws RemoteException;
    void event(Event e) throws RemoteException;
}
```

Protože RMI umožňuje jen volat metody server aplikace, je nutné, aby se applet aktivně dotazoval (polling) server aplikace, zdali má obdržet zprávu. Tato funkčnost bude zajištěna částí appletu pro periodické volání `getCommands()` a poté pokud jsou nějaké příkazy obdrženy, bude informovat applet.

4.2 Reprezentace vzdálených objektů

Vytvoření vzdálených objektů se bude provádět obecným mechanismem pomocí objektu Class, který umožní vytvořit instanci třídy, kterou objekt třídy Class popisuje. Podobně se bude postupovat při volání metod vzdálených objektů. Použije se objekt třídy Method, který popisuje metodu třídy. Zavolání popisované metody se provede metodou `invoke()`. Takže pro zavolání vzdálené metody nám bude stačit popis třídy, textový název metody, popis parametrů metody a hodnoty parametrů.

Příklad vytvoření instance a volání metody

```
// vyvtoření instance
Class jbutton = javax.swing.JButton.class;
javax.swing.JButton obj = (javax.swing.JButton)jbutton.newInstance();

// volání metody
Class[] parameterType = { String.class };
Object[] args = { ‘‘Hello World!’’ }
jbutton.getMethod(‘‘setText’’, parameterType).invoke(obj, args);
```

Protože nelze použít přímou referenci na vzdálený objekt, musíme si pomoct řetězcovou identifikací, která bude stejná pro klientský i serverový objekt. A na obou částech se bude tato identifikace převádět na kontrétní objekt.

4.3 Popis klienta

Applet bude vykonávat příkazy, které mu pošle server, jako třeba vytvoř tlačítko na pozici x,y a nastav popis na “abc”. Bude se tedy jednat o příkazy pro vytvoření a úpravu vzhledu grafického uživatelského rozhraní, ale také zde budou příkazy pro nastavení událostí, které mají být posílány zpět na server. Budou to události vznikající při nějaké akci uživatele, jako je kliknutí na tlačítko či jiné akci. Poslání události zpět na server bude provedeno prostřednictvým voláním vzdálené metody serveru `event()`. V parametru funkce bude předána událost `Event`.

Formát události:

```
class Event
{
    String srcObjId; // identifikace zdrojového objektu
    String evId; // identifikace události
    Class evDispClass; // typ objektu (doručovatele), který bude
                       // obsluhovat tuto událost
    Object[] evData; // hodnoty událostí
    List<UpdateCommand> updCommands; // synchronizační příkazy
}
```

Součástí události jsou i synchronizační příkazy, které generuje klient, aby během obsluhy události měl server k dispozici aktuální stav uživatelského rozhraní, jaký byl v době vzniku události na klientovi.

Formát synchronizačního příkazu:

```
class UpdateCommand
{
    String objId; // identifikace objektu
    String method; // název volané metody (jedno parametrová)
    Class  valClass; // typ hodnoty
    Object val; // hodnota
}
```

4.4 Popis serveru

Server přijímá zprávy od klienta prostřednictvím RMI a zařazuje si je do fronty zpráv, kde čekají na zpracování. Z fronty zpráv si zprávu vyzvedne virtuální grafické rozhraní a předá ji ke zpracování vlastní aplikaci. Aplikace během zpracování zprávy může prostřednictvím virtuálního grafického rozhraní generovat příkazy, které budou měnit uživatelské rozhraní. Tyto příkazy jsou vkládány do fronty příkazů. Seznam těchto příkazů je poslán přes RMI klientovi.

Základ pro tvorbu příkazů. Identifikuje objekt, kterému je příkaz určen.

```
class Command
{
    String objId; // identifikace objektu
}
```

Příkazy, které generuje server jsou:

CreateComponent – Vytvoří komponentu podle zadaného typu.

```
class CreateComponentCommand extends Command
{
    Class<?> objClass; // typ objektu, který má být vytvořen
}
```

AssociateComponent – Vytvoří vazbu mezi dvěma komponentami. Většinou se jedná o umístění jedné komponenty do jiné. Například `JButton` do `JPanel`. Ještě obsahuje popis metody, kterou má použít, například u `JPanel` má použít `add()` a `JScrollPane` má použít `setViewportView()`.

```
class AssociateComponentCommand extends Command
{
    String methodName; // název volané metody
    Class<?>[] parameterType; // typy parametrů volané metody
    Object[] parameters; // hodnoty parametrů volané metody
    String assocObjId; // identifikace nadřazené komponenty
}
```

SetAttributeComponent – Příkaz pro zavolání nějaké metody komponenty, obvykle se jedná o nastavení nějaké vlastnosti. Příkaz obsahuje identifikaci komponenty a popis metody, kterou chceme zavolat. Popis metody se skládá z názvu metody, typy parametrů metody a hodnoty parametrů, které chceme nastavit.

```
class SetAttributeComponentCommand extends Command
{
    String methodName; // název volané metody
    Class<?>[] parameterType; // typy parametrů volané metody
    Object[] parameters; // hodnoty parametrů volané metody
}
```

CreateModel – Vytvoří model, který využívají některé komponenty (JComboBox – ListModel, JTable – TableModel), podle zadaného typu.

```
class CreateModelCommand extends Command
{
    Class<?> objClass; // typ objektu, který má být vytvořen
}
```

AssociateModel – Vytvoří vazbu mezi komponentou a modelem. Obsahuje identifikaci komponenty a identifikaci modelu a popis metody komponenty, která se má použít.

```
class AssociateModelCommand extends Command
{
    String methodName; // název volané metody
    Class<?>[] parameterType; // typy parametrů volané metody
    Object[] parameters; // hodnoty parametrů volané metody
    String assocObjId; // identifikace nadřazené komponenty
}
```

SetAttributeModel – Příkaz pro zavolání nějaké metody komponenty, obvykle se jedná o nastavení nějaké vlastnosti. Příkaz obsahuje identifikaci modelu a popis metody, kterou chceme zavolat. Popis metody se skládá z názvu metody, typy parametrů metody a hodnoty parametrů, které chceme nastavit.

```
class SetAttributeModelCommand extends Command
{
    String methodName; // název volané metody
    Class<?>[] parameterType; // typy parametrů volané metody
    Object[] parameters; // hodnoty parametrů volané metody
}
```

SetEvent – Slouží pro nastavení odebírání některé události, která nastane nad zadanou komponentou. Příkaz obsahuje identifikaci komponenty, identifikaci `EventHandler`, který se má použít, a popis metody, která se má zavolat.

```
class SetEventCommand extends Command
{
    String methodName; // název volané metody
    Class<?>[] parameterType; // parametry volané metody
    String evObjId; // identifikace obsluhy události
    Class  evObjClass; // typ obsluhy události (pro vytvoření instance)
}
```

RegisterEvent – Slouží pro registraci událostí, které se mají posílat ke zpracování na server. Příkaz obsahuje identifikaci komponenty a identifikaci EventHandler.

```
class RegisterEventCommand extends Command
{
    String evObjId; // identifikace obsluhy události
}
```

ConfirmEvent – Slouží pro potvrzení, že událost, která byla poslána na server ke zpracování, už byla zpracována. Příkaz obsahuje identifikaci události, která byla zpracována.

```
class ConfirmEventCommand extends Command
{
    String evObjId; // identifikace obsloužené události
}
```

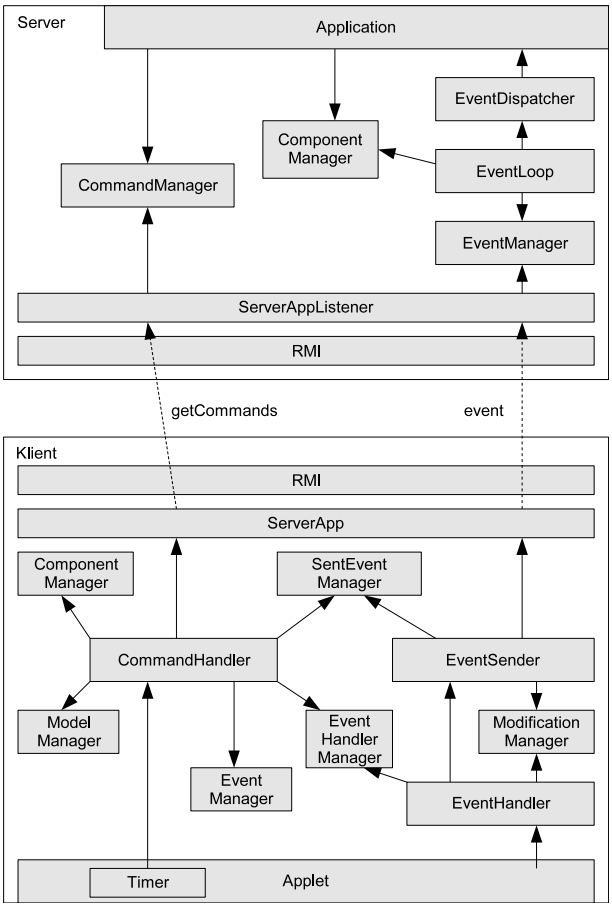
Ve výčtu generovaných příkazů jsou příkazy pro komponenty a pro modely, protože pro identifikaci komponent z knihovny AWT/Swing lze využít vlastnost `name`, kterou nabízí základní třída `Component`, od které jsou pak odvozeny veškeré prvky uživatelského rozhraní. Modely nemají žádnou vhodnou výchozí třídu, takže je nutné si odvodit vlastní třídy a sjednotit je implementací rozhraní `Identifiable`.

```
interface Identifiable
{
    String getId();
    void setId(String objId);
}
```

Kapitola 5

Realizace

Celková architektura navrženého systému je znázorněna na obrázku 5.1. Celý systém lze rozdělit na serverovou a klientskou část. V této kapitole se budeme věnovat podrobnějšímu popisu obou těchto částí.



Obrázek 5.1: Jádro sytému

5.1 Klient

Pomocí časovače si klient zajistí, aby periodicky prováděl získávání příkazů pomocí vzdáleného volání `getCommnads()`. Získané příkazy jsou vykonány (CommandHandler).

Události, které vzniknou nad uživatelským rozhraní, zpracovávají posluchači, kteří signalizují, že došlo ke změně uživatelského rozhraní a tuto změnu předají správci změn (ModificationManager). Také kontrolují zdali tato událost je registrována serverovou aplikací a pokud ano, tak generují událost, která bude prostřednictvím odesílatele událostí (EventSender) poslána na server. Odesílatel události k dané události ještě přiloží synchronizační příkazy od správce změn a pak ji odešle na server vzdáleným volání `event()`. Identifikaci odeslané události předá správci odeslaných událostí (SentEventManager) a on provede zablokování uživatelského rozhraní, aby uživatel nemohl vytvářet další události dokud tato událost nebude serverovou aplikací zpracována. Zpracování události je od serveru signalizováno potvrzovacím příkazem (ConfirmEvent).

RMI – Slouží pro volání vzdálených metod.

ServerApp – Poskytuje rozhraní serveru, přes toto rozhraní se provádí volání `event()` a `getCommands()`.

CommandHandler – Provádí zpracování příkazů, které obdrží od server aplikace. Typy příkazů, které zpracovává jsou:

- **CreateComponent** – Vytvoří komponentu podle zadané `Class` a přiřadí ji identifikaci, která je stejná jak u klientské komponenty tak i u serverové komponenty.
- **AssociateComponent** – Vytvoří vazbu mezi dvěma komponentami. Většinou se jedná o umístění jedné komponenty do jiné. Například `JButton` do `JPanel`. Ještě obsahuje popis metody, kterou má použít, například u `JPanel` má použít `add()` a `JScrollPane` má použít `setViewportView()`.
- **SetAttributeComponent** – Příkaz pro zavolání nějaké metody komponenty, obvykle se jedná o nastavení nějaké vlastnosti. Příkaz obsahuje identifikaci komponenty a popis metody, kterou chceme zavolat. Popis metody se skládá z názvu metody, typy parametrů metody a hodnoty parametrů, které chceme nastavit.
- **CreateModel** – Vytvoří model, který využívají některé komponenty (JComboBox – ListModel, JTable – TableModel), podle zadané `Class` a přiřadí ji identifikaci, která je stejná jak u klientského modelu tak i u serverového modelu.
- **AssociateModel** – Vytvoří vazbu mezi komponentou a modelem. Obsahuje identifikaci komponenty a identifikaci modelu a popis metody komponenty, která se má požít.
- **SetAttributeModel** – Příkaz pro zavolání nějaké metody komponenty, obvykle se jedná o nastavení nějaké vlastnosti. Příkaz obsahuje identifikaci modelu a popis metody, kterou chceme zavolat. Popis metody se skládá z názvu metody, typy parametrů metody a hodnoty parametrů, které chceme nastavit.
- **SetComponentEvent** – Slouží pro nastavení odebírání některé události, která nastane nad zadanou komponentou. Příkaz obsahuje identifikaci komponenty, identifikaci `EventHandler`, který se má použít, a popis metody, která se má zavolat.
- **RegisterEvent** – Slouží pro registraci událostí, které se mají posílat ke zpracování na server. Příkaz obsahuje identifikaci komponenty a identifikaci EventHandler.

ConfirmEvent – Slouží pro potvrzení, že událost, která byla poslána na server ke zpracování, už byla zpracována. Příkaz obsahuje identifikaci události, která byla zpracována. Součástí zpracování tohoto příkazu je odblokování uživatelské rozhraní, které bylo zablokováno před odesláním události na sever ke zpracování.

ComponentManager – Správce komponent uchovává seznam vytvořených klientských komponent a poskytuje převod identifikace na konkretní komponentu.

ModelManager – Správce modelů uchovává seznam vytvořených klientských modelů a poskytuje převod identifikace na konkretní model.

EventManager – Správce obsluh událostí uchovává seznam vytvořených obsluh událostí a poskytuje převod identifikace na konkretní obsluhu události.

EventHandlerManager – Správce registrovaných událostí obsahuje seznam registrovaných událostí, které mají být poslány na server ke zpracování.

SentEventManager – Správce odeslaných událostí obsahuje seznam odeslaných událostí na server ke zpracování, a které ještě nebyli potvrzeny, že jsou zpracovány.

Timer – Časovač slouží pro zajištění pravidelného opakovaní volaní získání a zpracování příkazů ze serveru.

Applet – Na appletu jsou umísťovány prvky uživatelského rozhraní, které jsou viditelné uživateli a se kterými může pracovat.

EventHandler – Obsluha vzniklé události. Pokud je tato událost registrovaná, tak se pošle ke zpracování na server. Pokud je to událost, která vnikla uživatelskou změnou některé komponenty, tak se tato změna uloží k pozdějšímu odeslání na server. Nejlépe jako součást nějaké události, která je poslána na server ke zpracování.

ModificationManager – Správce uživatelských změn některých komponent. Následně pak generuje příkazy pro synchronizaci stavu klientských a serverových komponent.

EventSender – Součást pro odesílání události na server. K události přiloží změny k synchronizaci, aby serverová aplikace měla k dispozici aktuální data ke zpracování události. Například vyplněný text v nějakém editovatelném textovém poli. Před odesláním události provede zablokování uživatelského rozhraní dokud nebude událost zpracována.

5.2 Server

Důležitá část je zde smyčka zpráv (EventLoop), která umožňuje tvořit událostmi řízenou aplikaci. Smyčka zpráv čeká na příchozí událost a až je nějaká událost signalizovaná, vyzvedne si ji u správce událostí (EventManager) a provede její zpracování tím, že nejdřív provede synchronizaci pomocí synchronizačních příkazů, které jsou součásti události, a pak událost předá zadanému doručovateli událostí. Který doručovatel událostí má být použit, je předáno ve struktuře `Event`. Přes doručovatele je událost předána cílovému objektu a jeho posluchačům. Během zpracování událostí aplikací můžou být generovány příkazy pro změnu uživatelského rozhraní. Příkazy jsou předávány správci příkazů (CommandManager), který je pak předá klientovi, až si o ně požádá vzdálením voláním `getCommands()`. Po ukončení obsluhy události musí smyčka zpráv poslat potvrzovací příkaz (ConfirmEvent), kterým informuje klienta, že tato událost byla zpracována.

RMI – Slouží pro příjem vzdálených volání z klienta.

ServerAppListener – Implementace rozhraní serveru.

CommandManager – Správce příkazů, které mají být odeslány na klienta.

ComponentManager – Správce komponent a modelů, které jsou vytvořeny. Poskytuje převod identifikace na konkretní komponentu či model.

EventManager – Správce událostí, které poslal klient ke zpracování.

EventLoop – Smyčka zpráv, která zajišťuje odebíraní příchozích událostí a předává je aplikaci ke zpracování. Před tím než předá událost aplikaci ke zpracování, tak provede synchronizaci podle přiložených příkazů.

EventDispatcher – Doručení události registrovaným posluchačům. Zajišťuje převod formátu doručené události na událost, která je pak předána posluchačům.

Application – Implementace libovolné aplikace, která využívá vzdáleně ovládané uživatelské rozhraní.

5.3 Stavba server knihovny

Knihovna pro tvorbu vzdáleně ovládaných aplikací je vytvořena tak, aby odpovídala knihovně AWT/Swing, takže je možné změnit pouze importované balíky z AWT/Swing na balíky se vzdáleně ovládanými komponentami. A tím bude převedena desktopová aplikace na vzdáleně ovládanou aplikaci.

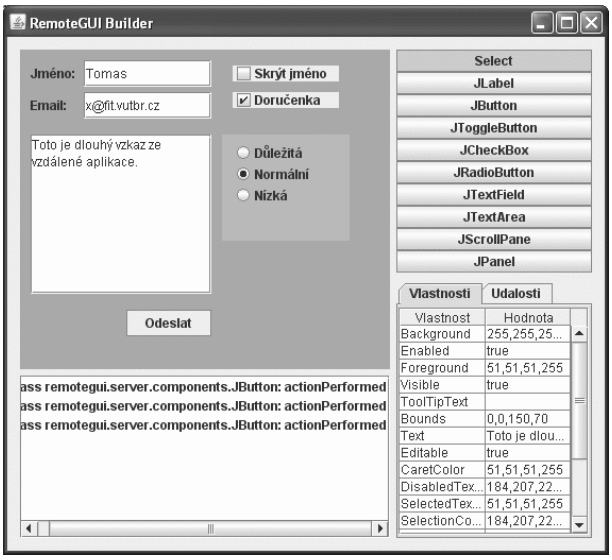
Hierarchie vzdáleně ovládaných komponent je v příloze na obrázku A.1.

Původně bylo myšleno, že pro reprezentaci událostí budou využity typy z knihovny AWT/Swing, ale protože některé typy událostí se omezují jen na zdrojový typ komponenty `java.awt.Component`, tak je nutné si tyto události implementovat a změnit typ zdrojové komponenty na obecný typ `Object`. Samozřejmě pokud jsou změněny typy událostí, je nutné implementovat pro tyto události i rozhraní posluchačů a adaptérů.

Kapitola 6

Interaktivní editor aplikace ovládané přes WWW

Pro demonstrování funkčnosti systému vzdáleného ovládání aplikace přes WWW je vytvořena aplikace (Interaktivní editor aplikace ovládané přes WWW), která bude demonstrovat funkčnost této knihovny. Vytvořená aplikace je interaktivní a veškeré změny nad uživatelském rozhraní se projeví v appletu na klientovi a naopak změny v appletu se také projeví v editoru (ve formuláři i v seznamu vlastností).



Obrázek 6.1: Interaktivní editor

6.1 Možnosti editoru

- Lze vytvářet prvky uživatelského rozhraní, které mají být zobrazeny v appletu na klientovi.
- Lze nastavovat jednotlivé atributy u každého prvku uživatelského rozhraní.
- Lze registrovat události, o kterých má klientský applet informovat editor.

Obrázek 6.2: Vytvořené vzdálené uživatelské rozhraní

- Veškeré uživatelské změny v appletu na klientovi se projeví i v editoru.

6.2 Části editoru

Panel komponent uživatelského rozhraní – Jednotlivé komponenty, které lze umístit do výsledného formuláře.

Tabulka s vlastnostmi a událostmi – Vlastnosti, které lze nastavit, a události, které si lze zaregistrovat. Změna atributu komponety se projeví i ve vzdálené komponentě.

Seznam signalizovaných událostí – Výpis zaregistrovaných událostí, které nastaly.

Panel pro umísťování prvků uživatelského rozhraní – Zde je vidět uživatelské rozhraní, které se zobrazuje u klienta.

K dispozici máme sadu komponent uživatelského rozhraní, které lze umístit do vytvářeného formuláře. U každého prvku uživatelského rozhraní máme možnost nastavovat některé jeho atributy a také registrovat události, o kterých chceme být informováni.

Seznam dostupných prvků uživatelského rozhraní:

JButton – běžné tlačítko

JRadioButton – přepínač, automaticky je umístěn do ButtonGroup

JCheckBox – zaškrtávací tlačítko

JToggleButton – stavové tlačítko (zmáčknuté, nezmáčknuté)

JTextField – vstupní jednořádkové textové pole

JTextArea – vstupní víceřádkové textové pole

JScrollPane – panel, který umožní zobrazit posuvníky a řídit tak vloženou komponentu

JPanel – kontejner na prvky uživatelského rozhraní

JLabel – statický textový popisek

6.3 Popis obsluhy editoru

Spuštění editoru

Editor lze spustit bez parametrů a poté bude poslouchat na všech síťových rozhraní a bude poslouchat na výchozím portu 3232. Editoru lze vnutit adresu a port, na kterém má poslouchat, stačí do parametru na příkazové řádce napsat třeba:

```
java remoteguibuilder.Main 147.229.66.44 5678
```

Spuštění appletu

Applet si je nutné přes internetový prohlížeč stáhnout právě z takové adresy, na které poslouchá i editor. Je to dáno bezpečnostním omezením appletu, který pak může navázat síťové spojení jen s počítačem, odkud byl tento applet stažen. Adresa, na kterou se má applet připojit, je předána jako parametr appletu. Tyto parametry jsou napsány v HTML stránce, ve které je umístěn i applet.

Příklad:

```
<applet code=‘‘remotegui/client/ClientApplet.class’’ width=400 height=300>
    <!-- nastavit vhodne parametry -->
    <param name=‘‘host’’ value=‘‘192.168.0.57’’ />
    <param name=‘‘port’’ value=‘‘3232’’ />
</applet>
```

Výběr prvku pro umístění na formulář

V panelu prvků si vybereme prvek, který chceme umístit na formulář a poté levým kliknutím ve formuláři se nám na zvolené pozici objeví vybraný prvek. Pokud chceme umístit některé komponenty do jiné komponenty jako třeba do JPanel nebo JScrollPane, tak to můžeme provést tak, že na komponentě, kam chceme prvek umístit, klikneme levým tlačítkem na myši.

Nastavení vlastností, událostí

V panelu prvků vybereme možnost “Select” a pravým kliknutím na požadovaný prvek se zobrazí v záložkách tabulka se seznamem vlastností, které lze změnit, a s událostmi, které si lze zaregistrovat, a pak editor bude o těchto událostech informován. Úprava libovolné vlastnosti se okamžitě projeví jak na formuláři tak i na vzdáleném uživatelském rozhraní.

Formát některých typů vlastností

Boolean : true/false

Barva : r,g,b,a (jednotlivé barvy se zadávají číslem od 0 do 255)

Pozice : x,y,w,h (jednotlivé údaje jsou celočíselné kladné hodnoty)

Interakce nad appletem

Vytvořené uživatelské rozhraní si lze ihned vyzkoušet v appletu na klientovi. Uživatelské změny se projeví zpět i v editoru, ale až pokud nastane nějaká registrovaná událost.

Kapitola 7

Závěr

Navržený a vytvořený systém pro tvorbu vzdáleně ovládaných aplikací je schopen fungovat jako webová aplikace s uživatelským rozhraním jako má běžná deskotopová aplikace. Protože struktura a formát knihovny komponent pro tvorbu vzdáleně ovládaných aplikací odpovídá standardní javovské knihovně AWT/Swing, lze použít kód již hotové aplikace s uživatelským rozhraním, vytvořeným pomocí AWT/Swing, a změnit pouze importované balíky. A máme hotovou vzdáleně ovládanou aplikaci. V příloze B je příklad implementace aplikace pomocí AWT/Swing a knihovny pro tvorbu vzdáleně ovládaných aplikací (RemoteGUI), který demonstruje jednoduchost převodu. Systém je použitelný i tam, kde je komunikace omezována firewall, protože komunikaci zprostředkovává RMI, které je schopno komunikaci zabalit do HTTP protokolu.

Stávající knihovna komponent zatím nabízí pouze základní komponety, ze kterých nelze vytvořit plnohodnotné uživatelské rozhraní, takže bude potřeba tuto knihovnu rozšířit o další komponenty (JDialog, JComboBox, JList, JMenu, ..) a přidat podporu pro správce rozvržení. Další možností rozšíření je vyřešení možnosti víceuživatelského přístupu k aplikaci (možné řešení je pro každého uživatele vytvořit jednu instanci programu nebo v rámci jednoho programu spustit další vlákno). Dále je potřeba testovat aktivitu klienta a serveru, v případě klienta jestli už aplikaci neukončil (potřeba posílat zprávy, že je klient stále aktivní), v případě serveru pokud není už k dispozici (nezdaří se stáhnout seznam příkazů). V obou případech je nutné danou aplikaci ukončit. Dále je potřeba rozšířit systém a další typy příkazů (RemoveAssociateComponent – zrušení vazby mezi komponentama, UnregisterEvent – zrušení registrace odebíraných událostí, DeleteComponent – smazání komponenty).

Literatura

[1] WWW dokument. Java(tm) remote method invocation specification. http://java.sun.com/j2se/1.5/pdf/rmi-spec-1.5.0.pdf.

[2] Brett Spell. *Java Programujeme profesionálně*. Computer Press, 2002. ISBN 80-7226-667-5.

[3] WWW stránky. Html — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=HTML&oldid=132283938.

[4] WWW stránky. Html 4.01 specification. http://www.w3.org/TR/html401/.

[5] WWW stránky. Jdk(tm) 5.0 documentation. http://java.sun.com/j2se/1.5.0/docs/.

[6] WWW stránky. A swing architecture overview. http://java.sun.com/products/jfc/tsc/articles/architecture/.

[7] WWW stránky. Web application — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Web_application&oldid=132148087.

Dodatek A

Přílohy

A.1 Rozdělení do balíků

Jednotlivé části celého systému jsou děleny do balíků.

remotegui.common – Obsahuje definice typu objektů, které jsou společné pro obě části jak klientskou tak serverovou (příkazy, události, rozhraní serveru).

remotegui.client – Obsahuje implementaci klientské části jádra systému pro vzdálené ovládání komponent.

remotegui.client.components – Obsahuje implementaci upravených komponent a implementaci posluchačů pro zpracování událostí.

remotegui.server – Obsahuje implementaci serverové části jádra systému pro vzdálené ovládání komponent.

remotegui.server.components – Obsahuje implementaci knihovny vzdáleně ovládaných komponent. Odpovídá balíku `java.awt` a `javax.swing`.

remotegui.server.components.text – Obsahuje komponenty pro tvorbu komponent pracujících s textem. Odpovídá balíku `javax.swing.text`.

remotegui.server.components.event – Obsahuje implementace událostí, posluchačů a adaptérů. Odpovídá balíku `java.awt.event` a `javax.swing.event`.

remotegui.server.components.dispatch – Obsahuje implementaci doručovatelů událostí.

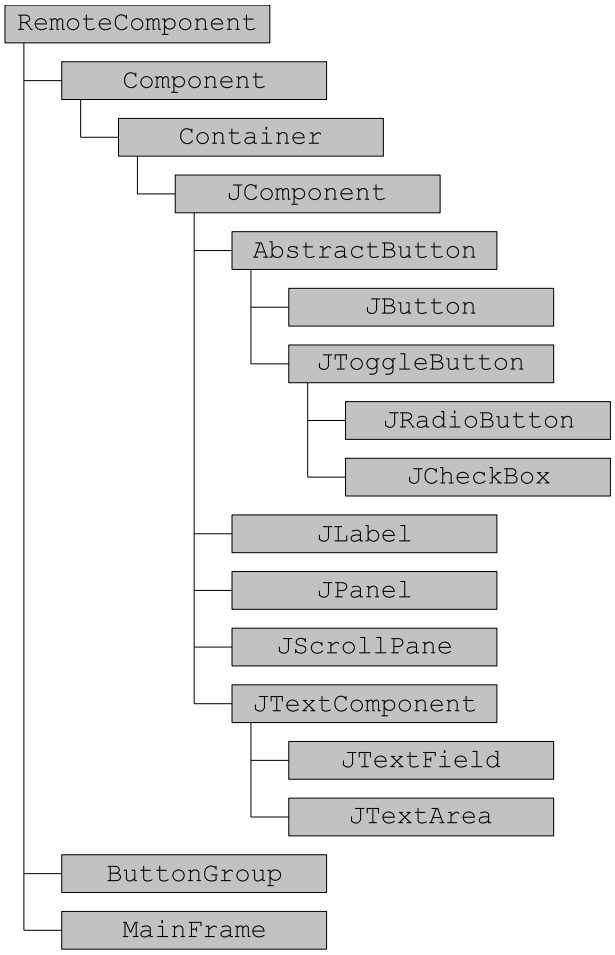
A.2 Popis knihovny komponent

Balík `remotegui.server.components` obsahuje komponenty pro tvorbu vzdáleně ovládaného uživatelského rozhraní.

RemoteComponent – Základ pro tvorbu vzdáleně ovládaných komponent. Poskytuje identifikaci a prostředky pro generovaní příkazů. Nemá ekvivalent na klientovi.

Component – Základní třída pro tvorbu komponent. Odpovídá `java.awt.Component`.

Container – Základní třída pro tvorbu komponent, které mohou obsahovat další komponenty. Odpovídá `java.awt.Contanier`.



Obrázek A.1: Hierarchie

JComponent – Základ pro tvorbu komponent, které mají odpovídat komponentám z knihovny Swing. Odpovídá `javax.swing.JComponent`.

AbstractButton – Základ pro tvorbu tlačítek. Odpovídá `javax.swing.AbstractButton`.

JButton – Běžné tlačítko. Odpovídá `javax.swing.JButton`.

JToggleButton – Tlačítko reprezentující stav zapnuto/vypnuto. Odpovídá komponentě `javax.swing.JToggleButton`.

JRadioButton – Tlačítko se stavem zapnuto/vypnuto. Vhodné pro tvorbu přepínačů. Odpovídá `javax.swing.JRadioButton`.

JCheckBox – Tlačítko se stavem zapnuto/vypnuto zobrazené jako zatrhávací rámeček. Odpovídá `javax.swing.JCheckBox`.

JLabel – Komponenta pro zobrazení statického textu. Odpovídá `javax.swing.JLabel`.

JPanel – Komponenta, do které je možné vkládat jiné komponenty a dělit tak uživatelské rozhraní do logických celků. Odpovídá `javax.swing.JPanel`.

JScrollPane – Komponenta pro zobrazení vertikálních a horizontálních posuvníků. Lze jimi ovládat jinou komponentu, která byla do této komponenty vložena. Používá se třeba v kombinaci s `JTextArea`. Odpovídá `javax.swing.JScrollPane`.

JTextComponent – Základní komponenta pro zobrazení editovatelného textu. Odpovídá `javax.swing.text.JTextComponent`.

JTextArea – Komponenta pro zobrazení víceřádkového editovatelného textu. Odpovídá `javax.swing.JTextArea`.

JTextField – Komponenta pro zobrazení jednořádkového editovatelného textu. Odpovídá `javax.swing.JTextField`.

MainFrame – Komponenta, která reprezentuje hlavní okno aplikace. Snaží se tvářit, že odpovídá `javax.swing.JFrame`, ale ve skutečnosti se převádí na objekt třídy `javax.swing.JApplet`, který na klientovi tvoří hlavní okno.

ButtonGroup – Komponenta pro tvorbu přepínačů. Odpovídá `javax.swing.ButtonGroup`.

Balík `remotegui.server.components.event` obsahuje třídy pro zpracování událostí ze vzdáleně ovládaného uživatelského rozhraní.

ActionEvent – Reprezentace události, která vznikla nějakou akcí (např. stisk tlačítka). Odpovídá `java.awt.event.ActionEvent`.

ActionListener – Rozhraní, které přijímá události `ActionEvent` ke zpracování. Odpovídá rozhraní `java.awt.event.ActionListener`.

CaretEvent – Reprezentace události, která vznikla změnou textového kurzoru. Odpovídá `javax.swing.event.CaretEvent`.

CaretListener – Rozhraní, které přijímá události `CaretEvent` ke zpracování. Odpovídá rozhraní `javax.swing.event.CaretListener`.

FocusEvent – Reprezentace události, která vznikla změnou aktivní komponenty. Odpovídá třídě `javax.swing.event.FocusEvent`.

FocusListener – Rozhraní, které přijímá události `FocusEvent` ke zpracování. Odpovídá rozhraní `java.awt.event.FocusListener`.

FocusAdapter – Implementace rozhraní `FocusListener`. Implementace odpovídá standardní třídě `java.awt.event.FocusAdapter`.

InputEvent – Reprezentace události, která vznikla nějakou vstupní akcí (stisk myši, stisk klávesy). Odpovídá `java.awt.event.InputEvent`.

KeyEvent – Obsahuje informace o události, která vznikla stiskem klávesy. Odpovídá `java.awt.event.KeyEvent`.

KeyListener – Rozhraní, které přijímá události `KeyEvent` ke zpracování. Odpovídá rozhraní `java.awt.event.KeyListener`.

KeyAdapter – Implementace rozhraní `KeyListener`. Implementace odpovídá standardní třídě `java.awt.event.KeyAdapter`.

MouseEvent – Obsahuje informace o události, která vznikla stiskem myši. Odpovídá `java.awt.event.MouseEvent`.

MouseListener – Rozhraní, které přijímá události `MouseEvent` ke zpracování. Odpovídá rozhraní `java.awt.event.MouseListener`.

MouseAdapter – Implementace rozhraní `MouseListener`. Implementace odpovídá standardní třídě `java.awt.event.MouseAdapter`.

Dodatek B

Ukázka použití knihovny

Ukázka aplikace s použitím knihovny AWT/Swing. Aplikace má zadaný text převést na text psaný velkými písmeny.

```
import java.awt.event.*;
import javax.swing.*;

public class AWTSwingApp
{
    public static void main(String[] args)
    {
        (new AppForm()).setVisible(true);
    }
}

import java.awt.event.*;
import javax.swing.*;

public class AppForm extends JFrame
{
    private JTextField text1;
    private JTextField text2;
    private JButton btn;

    public AppForm()
    {
        text1 = new JTextField();
        text2 = new JTextField();
        btn = new JButton();

        text1.setBounds(5, 5, 150, 20);
        text1.setText(‘‘hello world!’’);
        text2.setBounds(5, 60, 150, 20);
        btn.setBounds(30, 30, 100, 25);
        btn.setText(‘‘Convert’’);
        btn.addActionListener(new ActionListener()
```

```
        {
            public void actionPerformed(ActionEvent evt)
            {
                text2.setText( text1.getText().toUpperCase() );
            }
        });

        this.setLayout(null);
        this.add(text1);
        this.add(btn);
        this.add(text2);
    }
}
```

Obrázek B.1: Aplikace vytvořená pomocí AWT/Swing

Ta samá aplikace zapsaná pomocí knihovny RemoteGUI.

```
import java.net.InetAddress;
import remotegui.server.ServerAppListener;

public class RemoteGUIApp
{
    public static void main(String[] args)
    {
        String thisAddress = ‘‘’’;
        try
        {
            thisAddress = (InetAddress.getLocalHost()).toString();
        }
        catch(Exception e)
        {
        }

        int thisPort = 3232;
        ServerAppListener.create(thisPort);

        (new AppForm()).setVisible(true);
```

```
    }
}

import remotegui.server.components.event.*;
import remotegui.server.components.*;

public class AppForm extends MainFrame
{
    private JTextField text1;
    private JTextField text2;
    private JButton    btn;

    public AppForm()
    {
        text1 = new JTextField();
        text2 = new JTextField();
        btn = new JButton();

        text1.setBounds(5, 5, 150, 20);
        text1.setText(``hello world!'');
        text2.setBounds(5, 60, 150, 20);
        btn.setBounds(30, 30, 100, 25);
        btn.setText(``Convert'');
        btn.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent evt)
            {
                text2.setText( text1.getText().toUpperCase() );
            }
        });

        this.add(text1);
        this.add(btn);
        this.add(text2);
    }
}
```

Rozdíly jsou minimální, postačí jen změnit importované balíky a z deskotopové aplikace je aplikace ovládaná přes WWW.

Obrázek B.2: Vzdáleně ovládaná aplikace