VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

RODIČOVSKÝ MÓD V KDE

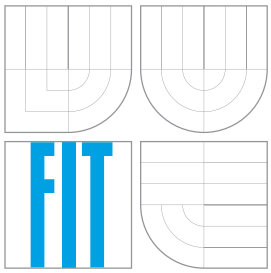
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PETR MRÁZEK

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

RODIČOVSKÝ MÓD V KDE

PARENTAL MODE IN KDE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — PETR MRÁZEK
AUTHOR

VEDOUCÍ PRÁCE — Ing. JOZEF MLÍCH,
SUPERVISOR

BRNO 2010

Abstrakt

Výtah (abstrakt) práce v českém jazyce.

Abstract

Výtah (abstrakt) práce v anglickém jazyce.

Klíčová slova

KDE Kiosk KAuth KAuthorized KioskTool autorizace autentizace PolicyKit

Keywords

KDE Kiosk KAuth KAuthorized KioskTool authorization authentication PolicyKit

Citace

Petr Mrázek: Rodičovský mód v KDE, bakalářská práce, Brno, FIT VUT v Brně, 2010

Rodičovský mód v KDE

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jozefa Mlícha. Další informace mi poskytli Jaroslav Řezník, Dario Freddi a Radek Nováček. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Petr Mrázek
17. května 2010

Poděkování

Děkuji autorům nástroje KDevelop4, protože bez něj bych se v kódu prostředí KDE tak rychle nezorientoval.

Obsah

Kapitola 1

Úvod

V moderních operačních systémech určených pro desktop (pracovní stanice, notebooky, etc.) se oproti předchozím verzím změnilo mnohé. Jednou takovou změnou je změna bezpečnostní politiky.

MS Windows postupně získal mnohem lepší zabezpečení ve verzích NT, kdy došlo k přechodu na nové jádro, a Vista, kde se poprvé objevila možnost jednoduše povolit obyčejnému uživateli přístup k nastavení systému pomocí UAC[1]. Zamezilo se tak nutnosti 'být administrátorem', což bylo do té doby výchozí nastavení po intalaci. Ve výsledku má uživatel povoleny akce, ke kterým by jinak přístup neměl.

Unixové a unixu podobné systémy se vyvíjely směrem opačným – od práv pevně svázaných s jeho účtem a skupinou, k mnohem volnějšímu pojetí. Za zmínku stojí například program sudo, který umožňuje uživateli spouštět programy s jinými právy než jsou jeho vlastní (převážně superuživatelská práva) případně program su, který je ještě staršího data.

Nově je k dispozici také PolicyKit ¿citace¡, který plní podobnou úlohu jako UAC na OS Windows. Aplikace využívající PolicyKit nemusejí mít pro změnu systémového nastavení jako je třeba datum a čas superuživatelská práva.

Prostředí KDE ¿citace¡ (po přeznačení KDE Software Compilation) je možné provozovat na více operačních systémech. Aplikace by tak musely používat řešení jako je UAC nebo PolicyKit, což by omezilo jejich přenositelnost. Tyto řešení tedy bylo potřeba nějak obalit. Za tímto účelem vzniklo rozhraní KAuth.

V rámci KDE však bylo možné již dřive měnit práva uživatelů – skrývat položky menu, zamezit použití terminálu a podobně. K tomu slouží Kiosk – kombinace vlastností několika systémů obsažených v KDE.

Tyto technologie a způsob jakým budou v práci využity jsou blíže popsány ve druhé kapitole. Třetí kapitola popisuje integraci systému KAuth do rozhraní KAuthorized a implementaci nástroje pro migraci. Čtvrtá kapitola se zabývá portem nástroje KioskTool do KDE4, úpravami v něm a návrhem nového grafického rozhraní.

[1] User Access Control

Kapitola 2

Základní technologie

V této kapitole popíši blíže zmíněné rozhraní a aplikace, tak aby bylo jasné jak fungují a jaké podmínky musí být splněny, aby bylo výsledné řešení ekvivalentem původního.

Dříve než se pustíme do samotného popisu technologií je třeba vymezit některé pojmy:

KAuth je rozhraní nad autorizačním řešením jako je například PolicyKit.

PolicyKit je starší verze autorizačního rozhraní dostupná v Linuxu.

PolicyKit1, nebo také "polkit"je jeho novější verze. Právě tu budu převážně používat.

polkit-qt je soubor knihoven obalující PolicyKit.

Authorization Services je obdoba PolicyKitu pro operační systém Apple OSX.

KConfig je systém pro ukládání a načítání nastavení v KDE.

Kiosk je souhrnný pojem pro některé vlastnosti KConfigu a postup načítání konfigurace obecně.

Kiosk profil je přiřazený uživateli nebo skupině uživatelů. Má vyšší prioritu než normální uživatelská nastavení KDE, ale nižší než globální systémové nastavení.

KAuthorized je tenké rozhraní nad KConfigem/Kioskem umožňující dotazování, zda jsou některé typy akcí povoleny.

KioskTool je aplikace pro správu Kiosk profilů. V KDE 3 umožňovala jednoduše nastavit některé aspekty Kiosku/KConfigu bez nutnosti měnit profily ručně.

2.1 Kiosk

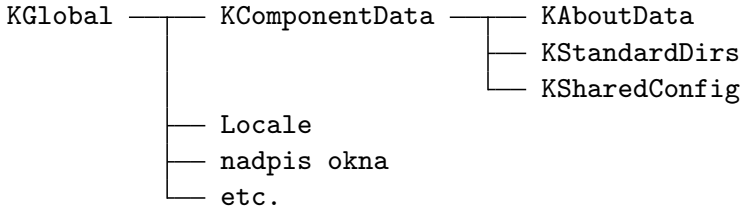
Zde a v podkapitole je čerpáno převážně z úvodu do Kiosku [3] a zdrojového kódu kdelibs.

Z pohedu administrátora Kiosk nabízí možnost upravit si KDE pro své vlastní účely. Například schovat některé položky v grafickém rozhraní aplikací, znemožnit uživatelům měnit nastavení a data aplikací nebo zamezit spuštění terminálu. Toho je docíleno nastevením některých částí konfigurace jako nezměnitelné (immutable) nebo vytvořením tzv. Kiosk profilů a přiřazením těchto profilů k uživatelům a skupinám. Tyto profily jsou složky obsahující v zásadě stejné soubory jako používá KConfig.

Kiosk je obecný pojem pro několik rozdílných vlastností systému KConfig a s ním provázanými částmi KDE. Nelze přesně určit jedno konkrétní místo ve zdrojovém kódu KDE, kde by bylo popsáno jakým způsobem funguje, ale podrobným zkoumáním lze vysledovat, jak je docíleno výsledného efektu. Z velké části je funkce Kiosku závislá na funkci KConfigu a způsobu jakým jsou načitány Kiosk profily během spouštění KDE aplikací a jejich komponent. Toto si blíže popíšeme, protože je to podstatné pro pochopení řešeného problému.

2.1.1 Načítání konfigurace během spuštění aplikace

Každá KDE aplikace má několik základních částí, obsažených ve struktuře KGlobal.



Obrázek 2.1: Struktura KGlobal

`KGlobal` je jmenný prostor obalující přístup k dalším komponentám a sdíleným zdrojům v rámci aplikace.

`KComponentData` obsahuje informace relevantní pro jednu komponentu aplikace. Platí, že aplikace může mít jednu hlavní komponentu.

`KAboutData` je soubor základních informace o programu – v případě hlavní komponenty jsou tyto informace použity také pro dialog "O Aplikaci"(About). Určuje název komponenty. Ten se použije pro načítání konfigurace a pro výběr složky s daty aplikace. Je nutno poznamenat, že aplikace může mít více než jednu komponentu.

`KSharedConfig` obsahuje sloučenou konfiguraci, efektivní pro program a zajišťuje, že konfigurace je sdílena mezi komponentami – šetří se tak pamětí a časem nutným k načtení konfigurace.

`KStandardDirs` slouží k určení, které složky budou použity jako zdroj dat a konfigurace pro aplikaci.

Právě `KStandardDirs` a `KSharedConfig` jsou relevantní pro popis načítání konfigurace. Během inicializace komponenty se nejdříve načítá obecná konfigurace. Ta obsahuje nastavení pro celé KDE a zmíněnou aplikaci, ovšem bez začleněných Kiosk profilů.

Potom je volána metoda `KStandardDirs::addCustomized`, která vyhledá Kiosk profily platné pro uživatele (na *nix systémech je soubor s jejich seznamem odkazován z '/etc/kde4rc') a zařadí je s prioritou (narozdíl od cest ke zdrojům určených v konstruktoru `KStandarDirs` jsou umístěny na začátek seznamu složek s konfiguračními zdroji).

Platí, že pokud má uživatel přiřazen svůj vlastní Kiosk profil, nezpracovávají se dále Kiosk profily pro skupiny v nichž je členem. V opačném případě, kdy uživatel nemá vlastní profil, přidají se s prioritou do cest v KStandardDirs všechny aplikovatelné skupinové profily. Pokud se počet cest s konfigurací změnil, je po návratu z addCustomized konfigurace znovu načtena.

2.1.2 KConfig

Kconfig je, jak již bylo řečeno, systém pro načítání a ukládání nastavení v KDE. V základu je navržen tak, aby bylo možné použít různé způsoby uložení nastavení, ale v praxi jsou použity .ini soubory (jsou jednoduché a rychle se načítají).

```
[$i]
[Colors]
CurrentPalette[$i]=Forty Colors

[ColumnMode]
FontWeight=50
PreviewSize=176

[Desktop Entry]
DefaultProfile=Shell.profile

[DetailsMode]
FontWeight=50

[Favorite Profiles]
Favorites=

[General][$i]
AutoExpandFolders=true
BrowseThroughArchives=true
ConfirmClosingMultipleTabs=false
FilterBar=true
FirstRun=false
RenameInline=true
ShowCopyMoveMenu=true
ShowFullPath=true
ShowSpaceInfo=true
ViewPropsTimestamp=2009,6,2,11,0,34
```

Výpis 2.1: Ukázka konfiguračního souboru KConfig

Na výpisu 2.1 vidět, jak je soubor členěn. Obsahuje skupiny a záznamy typu klíč-hodnota. Soubory se během načítání konfigurace slučují do jednoho konfiguračního celku (třída KConfig), podle pořadí ve kterém jsou zpracovány. To je určeno v

Celé soubory, skupiny jako je 'Directories' a jednotlivé záznamy jako je např. "AutoExpandFolders"je možné nastavit jako nezměnitelné – 'immutable' pomocí přidání značky [$i]. Jakmile je jednou nastavena na skupinu, soubor nebo hodnotu nezměnitelnost, jsou další objekty stejného typu při načítání ignorovány. V uvedeném příkladu je nastavena nezměnitelnost pro klíč CurrentPalette, skupinu General a celý soubor (nahoře). Máme-li tedy například pro uživatele 'test' nastaven Kiosk profil 'testprofil', který obsahuje konfigurační soubor pro aplikaci Akragator nastavený jako nezměnitelný, nebude moci uživatel nastavení tohoto programu změnit – obsah konfiguračního souboru v jeho domovské složce je ignorován.

2.1.3 KAuthorized

Toto rozhraní (přesněji jmenný prostor) je tenkým obalem nad KConfigem, který využívá všech jeho vlastností. Poskytuje KDE knihovnám a aplikacím možnost autorizace obecných akcí, KAkcí, konfiguračních modulů KControl, a přistupů k URL adresám. Chybí zde ovšem omezení přístupu ke zdrojům.

KAkce (třída `KAction` odvozená od `QAction` z knihovny Qt) a obecné akce jsou v rámci KDE činnosti, které může uživatel vykonat v aplikaci. Tyto akce mají název, který je v Kiosku použit pro jejich autorizaci a jsou často asociovány s nějakým ovládacím prvkem – položkou v menu apod.

Restrikce akcí jsou nastavovány v konfiguračních souborech ve skupině "KDE Action Restrictions". Zpravidla se jich používá ke schování prvků grafického rozhraní které jsou s akcemi provázány. Akce může být například otevření menu s nápovědou, změna pozadí na ploše nebo třeba vypnutí počítače z menu KDE. Pokud je taková akce zakázána pomocí Kiosku, prvky uživatelského rozhraní se nezobrazí (nevytvoří během inicializace aplikace).

Rozdíl mezi obecnou akcí a KAkcí v rámci `KAuthorized` je mizivý a funkce `AuthorizeKAction` je obal nad `authorize()`, který před název akce přidá prefix "action/". Příslušná funkce pak použije globální `KConfig` objekt aplikace k ověření, zda má uživatel tuto akci zakázánu (ve výchozím stavu jsou všechny povoleny).

Omezení přístupu ke zdrojům je nastavováno ve skupině `KDE Resource Restrictions` a je umístěno do globálního konfiguračního souboru jako je `kdeglobals`. Omezením zdrojů lze docílit toho, že aplikace "neuvidí"zdroje umístěné v uživatelově domovské složce. Lze tak zamezit například tomu, aby si uživatel aplikace rozšiřoval. I když je tato část Kiosku velmi podobná ostatním autorizačním funkcím v `KAuthorized`, není do něj přímo začleněna. Důvodem je, že je úzce provázána s třídou `KStandardDirs`, která tyto omezení zpracovává a používá pro vytvoření seznamů složek se zdroji. Seznam těchto omezení je potřeba znát ještě před tím, než je zcela načtena konfigurace z prostého důvodu – je možné omezit i zdroj pro konfiguraci. To by však nemělo bránit vytvoření nové funkce pro získání seznamu typu zdrojů s omezením přístupu v jmenném prostoru `KAuthorized` (mohla by například vracet `QStringList`).

Omezení přístupu k URL je nastavováno ve skupině `KDE URL Restrictions`. Tyto omezení jsou určeny pro zamítnutí přístupu k některým adresám URL. Vstupní parametry jsou seznam šestic hodnot a ověřovaná URL. Jednotlivé části je možné vynechat nebo místo nich použít proměnné.

```
rule_N=<action>,<referrer protocol>,<referrer host>,<referrer path>,
    <protocol>,<host>,<path>,<enabled>
```

Výpis 2.2: Syntaxe pro zápis omezení URL

Pravidla 1 a 2 jsou vypnuta, 3 - 6 zakazují otvírat a vypisovat soubory v $HOME a $TMP.

Po seznámení se s funkcí systému KAuth by z tohoto příkladu mělo být zřejmé, že neumí takováto data rozumně uložit. Omezeními na URL se tedy nebudu dále zabývat.

KDE používá třídu `KCModule` jako základ pro všechny moduly použité v aplikacích "Nastavení Systému", `kcmshell` a případně v jiných aplikacích, které používají KCM moduly pro úpravu svého nastavení. Za pomoci omezení přístupu ke KCM modulům je možné skrýt

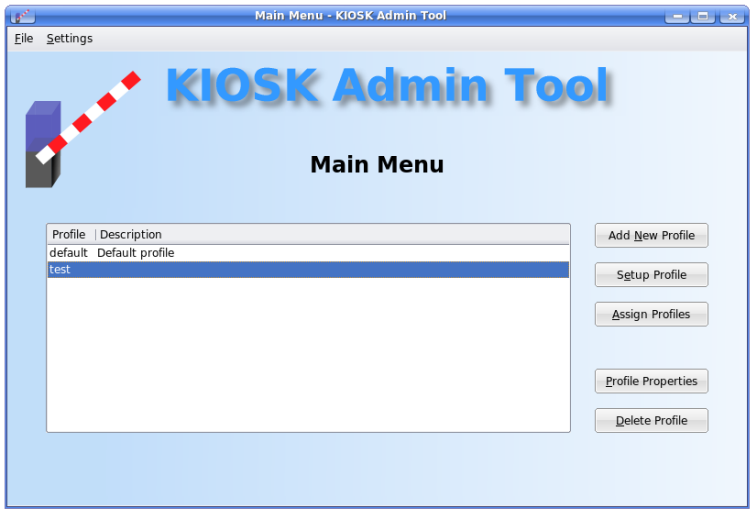
```
[KDE URL Restrictions][\$i]
rule_count=6
rule_1=open,,,,file,,,false
rule_2=list,,,,file,,,false
rule_3=open,,,,file,,\$HOME,true
rule_4=list,,,,file,,\$HOME,true
rule_5=open,,,,file,,\$TMP,true
rule_6=list,,,,file,,\$TMP,true
```

Výpis 2.3: Ukázka zápisu omezení URL

tyto moduly ve zmíněných aplikacích, aby je uživatel nemohl otevřít. To stejné funguje i pro služby KDE (KService). Omezení se vztahují k názvům těchto modulů – pokud budou moduly přejmenovány, přestanou omezení fungovat. V systému KConfig jsou uvedeny ve skupině 'KDE Control Module Restrictions'.

2.1.4 KioskTool

Aplikace KioskTool umožňovala v KDE3 spravovat některé specifické části Kiosk profilů pomocí grafického rozhraní. Nejprve popíši její funkčnost ve verzi pro KDE3 za pomocí obrázků, protože tak to bude nejlepší.



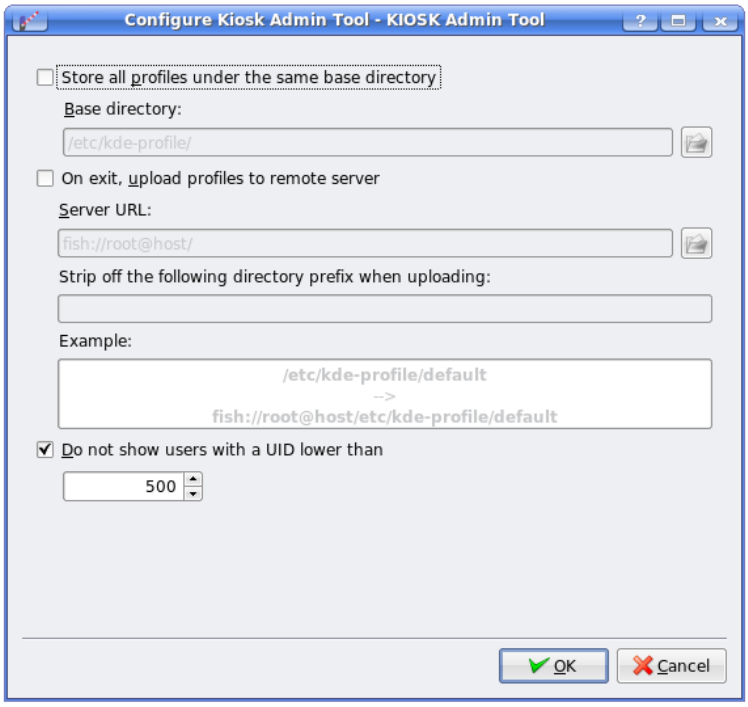
Obrázek 2.2: Úvodní obrazovka programu

Úvodní obrazovka 2.2 ukazuje seznam profilů s jejich popisem, lištu s menu a tlačítka pro manipulaci s profily. Velmi chybí možnost zkopírovat již existující profil.

Mezi nastavení programu 2.3 patří také možnost automaticky uložit profily na vzdálený serber.

Dialog pro vytvoření nového profilu 2.4 je poměrně jednoduchý. Umožňuje nastavit název a popis profilu a dále který uživatel bude vlastnit složku s profilem (bude mít přístup pro zápis) a kde bude profil uložen.

Při pohledu na obrazovku a dialogy pro přiřazení Kiosk profilů skupinám a uživatelům je nutné si připomenout, že když má uživatel přiřazen svůj vlastní profil, nevztahují se na



Obrázek 2.3: Dialogové okno s nastavením

něj profily pro skupiny kterých je členem. To program nijak nezvýrazňuje. Bylo by dobré, kdyby poskytoval nějaký pohled, který by ukazoval všechny profily efektivní pro uživatele.

Dialog pru úpravu profilu 2.6 je mohem zajímavější. Je možné z něj spouštět KCM moduly a jiné části systému KDE a tak docílit toho, že KioskTool může využít již existující funkcionalitu. Není proto nutné znovu 'vymýšlet kolo' a uživatel pro nastavení Kiosk profilů používá stejné nástroje jako pro normální nastavení. Tato původní verze nástroje KioskTool definuje všechny moduly a jejich vlastnosti v jednom velkém XML souboru.

Na obrázku 2.7 je vidět jak taková komponenta vypadá v akci. Uživatel může uzamčít obrázek na pozadí plochy a ukázat si náhled tohoto obrázku (náhled ve smyslu, že se pozadí plochy dočasně zamění). Zde by asi bylo lepší místo náhledu otvírat KCM modul pro nastavení plochy a případně nějaký režim pro rozvržení plasmoidů na ní. Zejména kvůli tomu, že v KDE4 je původně jednoduchá plocha nahrazena plochou plasma a tam je mnohem více věcí k nastavování.

I při krátké době nutné na seznámení s nástrojem (KioskTool 1.0 v Kubuntu 9.04) jsem narazil na fatální chyby, kdy se bez zjevného důvodu zhroutil. Nebude tedy možné se spolehnout na korektnost kódu.

2.2 PolicyKit

V této sekci jsem čerpal z manuálu [5], převážně pak z jeho částí [7] a [6].

PolicyKit je systém umožňující autentizovat uživatele, autorizovat akce těchto uživatelů a pomocí tzv. mechanismů tyto akce také provádět. Je to systém velmi modulární a přenechává implementaci mnoha těchto funkcí zásuvným modulům a programům které

Obrázek 2.4: Dialog pro vytvoření nového profilu

jej využívají. Pro každý typ rozšíření PolicyKitu existuje knihovna. Jednotlivé komponenty pak spolu komunikují pomocí IPC mechanismu D-Bus (ať už přímo nebo prostřednictvím knihoven).

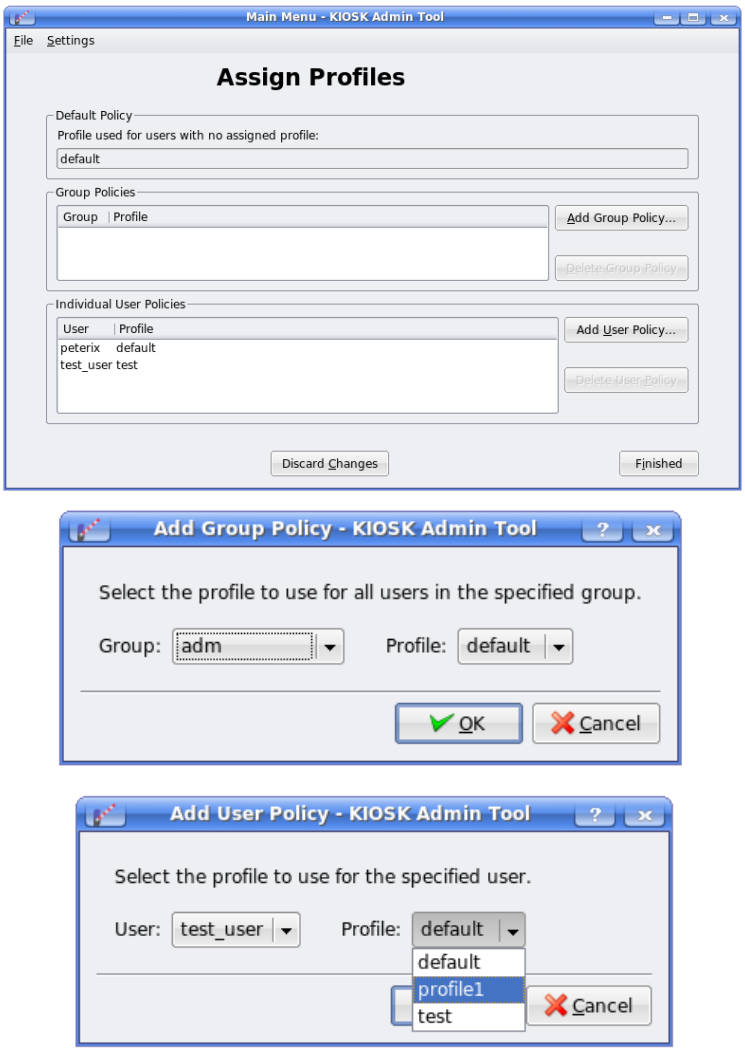
Mezi nejzákladnější části tohoto systému patří démon polkitd, implementující část PolicyKit Authority. Ta se stará o uložení databáze možných akcí a povolení a slouží jako centrální prvek celého systému.

Další částí, implementovanou prostředími jako je Gnome nebo KDE je tzv. autentizační agent. To je program, který například uživateli zobrazí okno pro zadání hesla, pokud je potřeba ověřit jeho totožnost. Pro samotné ověření je možné použít pomocný program běžící se superuživatslkými právy a používající k ověření systém jako je PAM – ten je dobře popsán v článku [1]. To ale není striktně vyžadováno. Lze i jen zobrazit obyčejné okno s dotazem typu Ano/Ne.

Třetí komponentou je 'mechanismus' nebo 'pomocník' – zde se jedná o službu pro vykonání privilegovaných akcí místo uživatele, který o provedení akce žádá. Čtvrtou a poslední komponentou je klient. To je program, který využívá služeb systému PolicyKit.

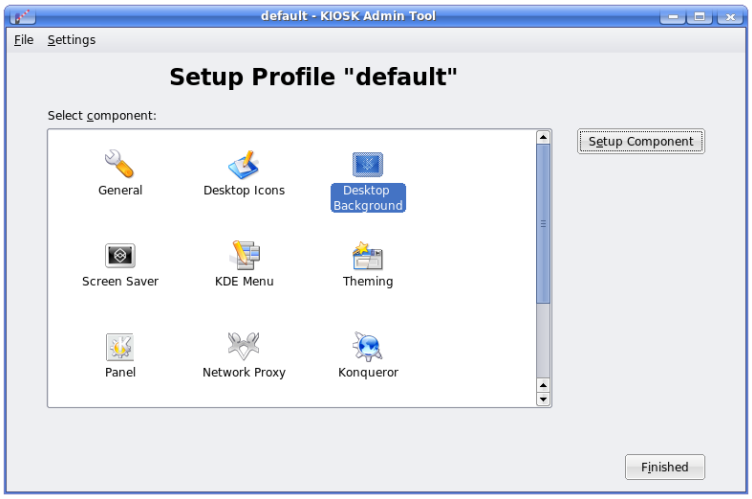
Průběh akcí v systému pro volání funkce pomocníka může vypadat například takto:

- Uživatel se přihlásí, je nastartováno sezení KDE a s ním i autentizační agent
- Autentizační agent se registruje u PolicyKit autority
- Uživatel spustí program, který využívá služeb PolicyKitu
- Program zavolá funkci pomocníka přes systém DBUS
- DBUS démon nastartuje program pomocníka, pokud již neběží (pomocník musí být v systému D-Bus registrován)
- Spuštěný pomocník se zeptá PolicyKit autority, jestli je volaná akce autorizována

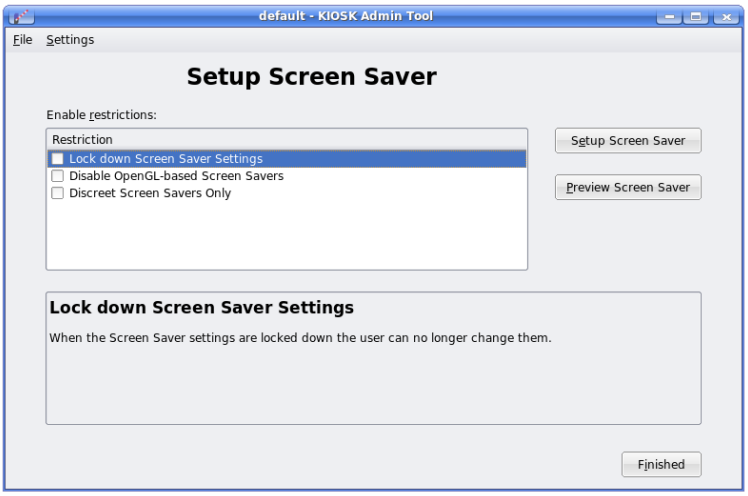


Obrázek 2.5: Dialogy pro přiřazení profilů

- PK autorita zkontroluje, jestli je již tato akce autorizována (autorizace může mít například platnost pro celé sezení). Pokud není zatím autorizována, zjistí jakým způsobem se má dosáhnout autorizace.
- Pokud je to nutné, zavolá PK autorita autentizačního agenta. Zobrazí se například okno, kde má uživatel zadat své login údaje. Je možné ověřovat buď pouze identitu uživatele, kdy stačí, že se přihlásí pod vlastním účtem, nebo je možné vyžadovat přihlášení účtu se superuživatelskými právy.
- Uživatel se přihlásí. Záleží na tom jakým způsobem to agent dělá – výchozí je PolicyKit mechanismus který obaluje PAM.
- Agent oznámí autoritě jestli byla autentizace úspěšná.
- Autorita vrací výsledek autorizace a uloží si ho do cache po dobu její platnosti.
- Pokud byl proces autorizacu úspěšný, pomocník provede požadovanou akci.



Obrázek 2.6: Dialog s nastavením profilu



Obrázek 2.7: Dialog jedné komponenty nástroje

- V tomto bodě může pomocník notifikovat program o výsledku akce.

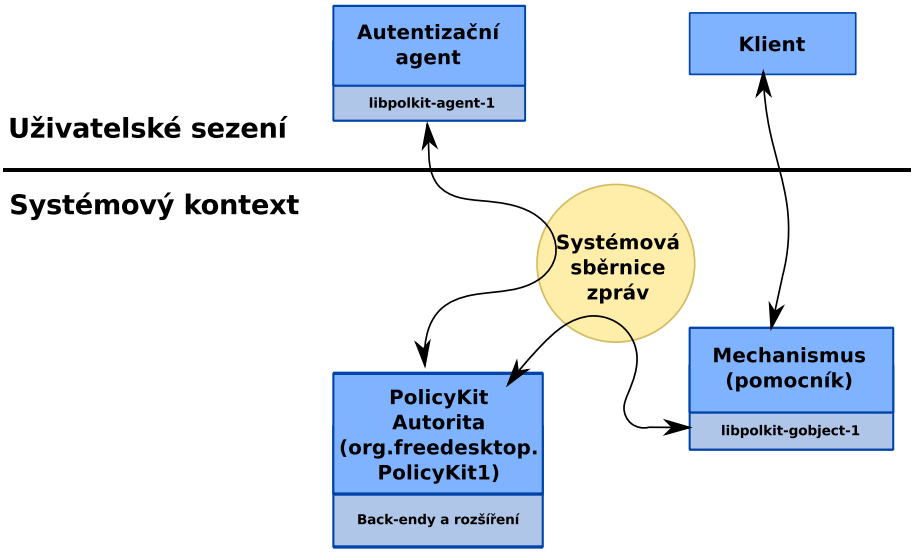
Toto samozřejmně není jediný možný průběh. Vše záleží na implementaci jednotlivých komponent. Jak jsem již uvedl, PolicyKit je velmi modulární systém a většinu jeho částí lze nahradit nebo rozšířit.

Nad PolicyKitem je postavena knihovna polkit-qt. Poskytuje v zásadě stejné funkce jako PolicyKit a lépe je integruje do prostředí knihoven QT.

Polkit-kde je nadstavbou nad knihovnou polkit-qt, která implementuje autentizačí agent pro prostředí KDE a KCM modul pro úpravu akcí povolených pro uživatele a skupiny.

Zatím jediným způsobem uložení povolených akcí v PolicyKitu je tzv. lokální autorita. Je to výchozí implementace PolicyKit Autority a využívá lokálně uložených textových souborů s příponou .policy a .pkla.

Nejdříve příklad 2.4 .policy souboru. Tyto soubory používají formát XML a vymezují seznam známých akcí. Instalují se do systému jako součást balíčků.



Obrázek 2.8: Architektura systému PolicyKit, přeloženo z [7]

Značka vendor určuje k čemu tento soubor akcí patří. V tomto případě je to KControl modul pro nastavení data a času. Soubor má dále nastavenu ikonu a samotný seznam akcí. Akce má popis (description), zprávu pro případ neúspěšné autorizace (message) a výchozí nastavení autorizace. PolicyKit rozlišuje mezi tzv. aktivním a pasivním sezením. Aktivní je například normálně přihlášený uživatel s vlastním X serverem. Pasivní může být sezení spuštěné dodatečně z login terminálu (su - username). Normálně může být aktivní pouze jedno sezení (to s kterým uživatel zrovna přímo pracuje). Soubor povoluje měnit nastavení času pouze uživateli a aktivním sezením, který se prokáže jak superuživatel.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE policyconfig PUBLIC
"-//freedesktop//DTD PolicyKit Policy Configuration 1.0//EN"
"http://www.freedesktop.org/standards/PolicyKit/1.0/policyconfig.dtd">
<policyconfig>
<vendor>Date and Time Control Module</vendor>
<icon_name>preferences-system-time</icon_name>
   <action id="org.kde.kcontrol.kcmclock.save" >
      <description>Save the date/time settings</description>
      <message>System policies prevent you from saving the date/time
          settings.</message>
      <defaults>{\"o}
         <allow_inactive>no</allow_inactive>
         <allow_active>auth_admin</allow_active>
      </defaults>
   </action>
</policyconfig>
```

Výpis 2.4: Ukázka souboru s definicí akce (.policy soubor)

Takto hrubě vymezená práva a omezení je dále možné pozměnit v souborech .pkla. Efektivní seznam povolených akcí se získá služením těchto souborů. Je tedy možné mít například jeden soubor s nastavením instalovaný z distribučního balíčku a druhý s vyšší

prioritou vytvořený administrátorem.

Obrázek 2.9: Struktura složek pro PolicyKit Local Authority s několika .pkla soubory. Převzato z [6]

```
/var/lib/polkit-1
└── localauthority
    ├── 10-vendor.d
    │   └── 10-desktop-policy.pkla
    ├── 20-org.d
    ├── 30-site.d
    ├── 50-local.d
    ├── 55-org.my.company.d
    │   └── 10-org.my.company.product.pkla
    └── 90-mandatory.d

/etc/polkit-1
└── localauthority
    ├── 10-vendor.d
    │   └── 01-some-changes-from-a-subvendor.pkla
    ├── 20-org.d
    ├── 30-site.d
    ├── 50-local.d
    ├── 55-org.my.company.d
    │   └── 10-org.my.company.product.pkla
    └── 90-mandatory.d
```

Local Authority pro ně zavádí poměrně propracovanou strukturu **??**. Pořadí načítání je určeno lexikografickým řazením složek a souborů v nich. Pokud je stejně pojmenovaný soubor v obou umístěních (ve /var/lib/ i /etc/), nejdříve se zpracuje ten ve /var/.

Ve struktuře **??** by pořadí zpracování vypadalo takto:

- 10-desktop-policy.pkla
- 01-some-changes-from-a-subvendor.pkla
- 10-org.my.company.product.pkla (z /var)
- 10-org.my.company.product.pkla (z /etc)

Na výpisu 2.5 je ukázka .pkla souboru. Popíšu na něm, jakým způsobem se získá výsledek autorizace určité akce. Uživatelé petr a pavel jsou členy skupiny zamestnanci.

Záznamy se zpracovávají tak jak jdou za sebou a platí, že zpracování pokračuje i po úspěšném porovnání uživatelkého jména/skupiny s těmi od žadatele. Jako příklad si vezmu uživatele petr, který požádá o autorizaci akce "`com.example.uzasnyprodukt.uzasnaakce`". Zaměstnanci mají tuto akci obecně povolenu bez nutnosti se přihlašovat v první skupině souboru. Zpracování však pokračuje a uživatel 'petr' se bude muset pro úspěšnou autorizaci akce přihlásit jako administrátor.

```
[Povolene akce pro zamestnance]
Identity=unix-group:zamestnanci
Action=com.example.uzasnyprodukt.*
ResultAny=no
ResultInactive=no
ResultActive=yes

[Zakazy pro par cernych ovci]
Identity=unix-user:petr;unix-user:pavel
Action=com.example.uzasnyprodukt.*
ResultAny=no
ResultInactive=no
ResultActive=auth_admin
```

Výpis 2.5: Ukázka souboru s nastavením PK Local Authority (.pkla soubor), Přeloženo z [6]

Rozdíl mezi položkami `ResultAny`, `ResultInactive` a `ResultActive` se nemusí zdát zjevný. `ResultActive` a `ResultInactive` jsou výsledky akce pro aktivní a neaktivní sezení – podobně jako značka `allow_active` a `allow_inactive` v .policy souborech. `ResultAny` pak určuje výsledek pro oba typy sezení. Alespoň jedna z těchto položek musí být přítomna. Hodnota položky udává jakým způsobem bude uživatel žádající o autorizaci akce autentizován.

2.3 KAuth

KAuth je nové rozhraní pro autorizaci v KDE. Je postaveno na již existujících rozhraních v operačních systémech. V OS na bázi Linuxu je to zpravidla PolicyKit, v OSX pak framework Authorization Services. Podporu pro nová rozhraní je možné přidat pomocí zásuvných modulů.

Pomocí KAuthu je možné autorizovat akce uživatele. Zásuvný modul se stará o veškeré ověřování akcí. Z pohledu PolicyKitu je zde aplikace používající KAuth v roli klienta.

Další funkce – a možná nejdůležitější a nejlépe fungující – poskytovaná KAuthem je vytvořit pokud možno co nejjednodušší program, který pak bude v odpověď na autorizovanou akci spuštěn příslušným autorizačním rozhraním. Takovýto program se nazývá KAuth pomocník (helper). Pomocník a aplikace která ho využívá mohou do jisté míry komunikovat oběma směry a je také možné sledovat průběh dlouho trvající akce uvnitř pomocníka. KAuth pomocník je rozšířením PolicyKit pomocníka o tyto zjednodušené komunikační funkce. Hlavní výhodou je, že za použití pomocníka lze provádět privilegované akce bez nutnosti být přihlášen pod administrátorským účtem nebo tohoto účtu používat ke spuštění aplikace jako celku.

Další službou KAuthu je registrace pomocníků a akcí v jeho jednotlivých zásuvných modulech. Uživatel KAuthu tedy specifikuje, jaké akce a pomocníky chce použít a při kompilaci budou tyto specifikace převedeny do formy srozumitelné pro jeho zásuvné moduly - např. PolicyKit nebo OSX Authorization Services.

KAuth nepodporuje provádění změn v seznamu platných akcí po kompilaci. Také neumožňuje měnit autorizované uživatele a skupiny pro akce – to je zcela přenecháno systémům pro autorizaci, které využívá. V případě PolicyKitu je toto nastavní realizováno v KControl modulu, který používá přímo knihovnu polkit-qt. Další nepodporovanou částí

je jakákoliv idea bezpečnostních profilů přiřazovaných uživatelům a skupinám. Nic takového konec konců není ani v PolicyKitu, který byl KAuthu modelem. Toto jsou celkem závažné nedostatky, které budu muset nějak vyřešit.

Největším omezením je však omezení na názvy KAuth akcí. Od rozhraní nad kterými funguje zdědil všechna jejich omezení v tomto směru. Pro jména akcí je tedy možné použít jenom malá písmena anglické abecedy, číslice a tečku jako oddělovač. Specifikace akcí v .actions souborech je také dále omezená. Všechny akce musejí mít společný základ (jmenný prostor).

Uvedu příklad .actions souboru 2.6. Tento byl použit k vygenerování podobného souboru pro PolicyKit, který byl uveden dříve zde: 2.4.

```
[Domain]
Name=Date and Time Control Module
Icon=preferences-system-time

[org.kde.kcontrol.kcmclock.save]
Name=Save the date/time settings
Description=System policies prevent you from saving the date/time settings.
Policy=auth_admin
Persistence=session
```

Výpis 2.6: Ukázka KAuth .actions souboru

Soubor obsahuje skupinu Domain, která popisuje ke které aplikaci patří (V tomto případě KCM modul pro nastavení data a času) a jakou má mít ikonu. Textové položky mohou být lokalizovány a velkou většinu jsem zde vynechal.

Po skupině domain následují definované akce. Platí, že název akce je také názvem skupiny v souboru. Název akce se skládá ze dvou částí – jmenného prostoru a názvu akce v něm. Zde je jmenným prostorem část "`org.kde.kcontrol.kcmclock`". Název akce je "`save`". Skupina dále obsahuje srozumitelný název (`Name`), popis (`Description`) a výchozí chování při pokusu o autorizaci.

Položka Policy určuje průběh autorizace. V případě, že je nastavena na hodnotu `yes`, je autorizace okamžitá. `no` naopak znamená, že akce nemůže být autorizována. Pokud je nastavena na `auth_self`, bude akce autorizována pokud se uživatel přihlásí pod svou vlastní identitou – ověří se tak, že je to opravdu on. Konečně `auth_admin` znamená, že akce bude autorizována pokud se uživatel přihlásí jako administrátor.

Poslední položkou skupiny je `Persistence`. Ta je nepovinná a udává, na jak dlouho bude autorizace udělena. Možnostmi jsou `session`, kdy bude platit dokud se neodhlásí a `always`, kdy nebude omezena.

Kapitola 3

Integrace KAuth do KAuthorized

První praktickou částí projektu je zjistit jak nejlépe integrovat systém KAuth do rozhraní KAuthorized. Toto není triviální úkol a nějakou dobu mi trvalo, než jsem přišel na to, jak toho alespoň částečně dosáhnout. Začnu popisem dvou variant, které jsem musel pro závažné nedostatky zavrhnout (postupně jak jsem analyzoval jednotlivé využitelné technologie). Třetí variantu, kterou budu implementovat, popisuji ve třetí sekci kapitoly.

Když nepřihlédnu k omezení KAuthu a budu ho brát jako ideální systém pro autorizaci uživatelských akcí, vypadá vše poměrně dobře. Na jedné straně Kiosk, podporující uživatelské a skupinové profily, změnu těchto profilů (stačí být superuživatelem a upravovat profily standardními nástroji KConfigu).

Na druhé straně zatím neznámý autorizační systém. Ze zadání je však možno vyčíst, že jejich sloučení je možné. Dá se tedy předpokládat, že takový systém bude umět přidávat nové restrikce a efektivně s nimi pracovat, dále že bude mít nějakou podporu pro zmíněné uživatelské a skupinové profily. Ideální by bylo, kdyby šlo specifikovat, jakým způsobem se budou slučovat a jaké tedy budou konečné výsledky.

Můj první a také nejjednodušší nápad byl vytvořit statický .actions soubor v kombinaci s integrací KAuth přímo do KAuthorized. V zásadě by toto mohlo fungovat a je to částí řešení. Není to ale celé řešení.

Problémem je zde to, že neexistuje definitivní seznam možných restrikcí akcí a zdrojů v Kiosk profilech. Toto by šlo řešit tak, že každý program KDE by tyto své akce a zdroje specifikoval v .actions souboru k tomu určeném a zahrnul by jeho překlad do svého sestavení. To ovšem vylučuje jakoukoliv možnost stejné úrovně podpory pro aplikace, které by takto akce nespecifikovaly.

Druhým problémem je neexistence možnosti nastavit autorizované akce pro uživatele a skupiny pomocí KAuthu. Nejen že nejdou nastavit, KAuth navíc nemá ani žádné ponětí o něčem, co by alespoň vzdáleně připomínalo způsob jakým se aplikují Kiosk profily. V případě použití PolicyKitu tyto vcelku základní funkce nemá ve formě knihovny žádná vrstva - PolicyKit, polkit-qt, polkit-kde ani KAuth.

Druhý nápad byl takovýto: když KAuth nepodporuje nic z toho, co by korektní implementace vyžadovala, bude nutné to nějak obejít, a to co nejjednodušším způsobem.

Možným řešením je jednoduše se vzdát toho, že budeme mapovat akce mezi Kioskem a KAuthem jedna ku jedné. Mělo by být přece možné umístit do KAuth pomocníka instanci KConfigu, která načte požadovaný Kiosk profil a bude přes systém D-Bus umožovat dotazování se na jednotlivé hodnoty v nich. Bylo by také možné tyto hodnoty měnit. V podstatě

by to byl stále pouze KConfig, jen obalen v pomocníkovi.

Problémy tohoto řešení nemusejí být zcela zjevné, pokusím se je ale popsat. Za prvé – nezíská se tím vůbec žádná výhoda z pohledu bezpečnosti. Kiosk profily musejí stále zůstat čitelné pro všechny uživatele, jinak by se při spuštění programů nemohly načíst a části které nebudou takto integrovány v pomocníkovi (vše s výjimkou omezení akcí a zdrojů) by tím by byly zcela neefektivní.

Odpadá tak výhoda KAuthu, kde jak seznam restrikcí, tak jaké restrikce pro koho platí může být skryt před uživateli. Například při použití PolicyKit Local Authority je možné nastavit všechny soubory čitelné jenom superuživatelem.

Za druhé – KAuth pomocníci mají omezenou životnost. Pokud nejsou využíváni po dobu deseti vteřin, jsou ukončeni (viz. zdrojový kód KAuth). Znamená to znovu načíst celý KConfig objekt. I když je toto načítání vysoce optimalizováno, stále je vcelku zbytečné načítat celou konfiguraci dvakrát - jednou v programu, který by volal KAuthorized a podruhé v KAuth pomocníkovi. Sdílet zde jeden objekt například umístěním do sdílené paměti by byl holý nesmysl, takže se taková věc ještě více prodraží – použijeme až dvakrát více paměti.

Jedna z vlastností KAuthu, kterou by se tak nedalo využít jsou srozumitelné názvy akcí v systému. Místo `org.kde.kiosk.action.help` a mnoha dalších by byly v systému jen akce pomocníka. V tom případě ne nedá mluvit o integraci KAuthu a KAuthorized.

Jsou tu i další problémy – KAuth podporuje použití pomocníků pouze v kombinaci se systémy PolicyKit a DBUS. Uživatelé Windows a OSX mají smůlu. Tomu se asi nevyhneme.

Je vidět, že ani tudy cesta nevede. Jak tedy postupovat když se zadání zdá být nesplnitelné?

3.1 Možné řešení

Řešení nebude až tak pěkné jak by mohlo být, kdyby KAuth podporoval zápis autorizací jakožto knihovní funkci. Stanovím zde některé základní podmínky, které integrace KAuth do KAuthorized musí splňovata omezení, která jsou daná implementací KAuthu. Některá je možné obejít, jiná bohužel ne.

Použití KAuth by nemělo být povinné a mělo by být zachováno chování rozhraní `KAuthorized` pokud možno tak, aby se z pohledu aplikací nic nezměnilo. Zde je problém to, že administrátor může, ale také nemusí nastavit v systému `KConfig` kteroukoliv položku jako nezměnitelnou. Pokud vložím kontrolu oprávnění pomocí KAuthu před kontrolu pomocí KConfigu, musím nějak zaručit, že omezení známá v KAuthu mohou být brána jako změnitelná, tak aby mohly být dále modifikovány pomocí nastavení v KConfigu. Pokud vím, KAuth ani PolicyKit toto neumožňují. Musí se také počítat s tím, že KAuth nemusí být funkční (je v nové verzi použité v této práci stále ještě nestabilní) nebo že konvertor zatím nikdy nebyl spuštěn a v těchto případech použít normální nastavení Kiosk profilu přes `KConfig`.

Dále je potřeba implementovat konvertor Kiosk profilů na nastavení lokálního systému pro autorizaci. Zde se budu muset držet PolicyKitu, protože jediný další podporovaný systém je Authorization Services v operačním systému Apple OSX. Hardware potřebný pro legální provoz OSX bohužel nemám. Takovýto konvertor by měl umět běžet jak KAuth pomocník i jako samostatný program spustitelný uživatelem z příkazové řádky. Konverze

z Kiosk profilů bude jednosměrná, bude používat polkit-qt pro získání seznamu podporovaných omezení a bude produkovat soubory .pkla, použitelné jen a pouze v PolicyKit Local Authority. Konvertor by mělo být možné, až bude stabilní, integrovat do KAuthu nebo balíku policykit-kde.

Dalším omezením konvertoru je, že pokud bude použita jiná autorita než PolicyKit Local Authority, konverze s ní nebude fungovat. Alse vzhledem k tomu, že tato autorita je výchozí, předpokládám, že bude k dispozici.

V konvertoru je nutné zohlednit to, že postup aplikace profilů v Kiosku je jiný než postup ověření autorizace v PolicyKitu. Musí se 'nasimulovat' Kiosk a jeho zvláštnosti, aby se nezměnilo chování KAuthorized.

Integrace KAuth do KAuthorized opět není triviální záležitostí – je to sice otázka několika volání funkcí KAuth, ale autorizační funkce KAuthu narozdíl od systémů na kterých staví nevrací chybové kódy v případě, že akce neexistuje. Toto bude potřeba změnit.

Omezení na zdroje (KDE Resource Restrictions) si vyžádají změny ve třídě `KStandardDirs` – bude potřeba umístit do jmenného prostoru `KAuthorized` funkci, která bude vracet seznam typů zdrojů s nastaveným omezením.

Je také třeba zopakovat, že KAuth/PolicyKit má některá omezení na názvy akcí. Všechny akce v jednom .actions souboru musejí mít společný jmenný prostor a v PolicyKitu soubor musí být podle tohoto jmenného prostoru pojmenován. Navíc mohou názvy obsahovat pouze malá písmena latinky, číslice a tečku jako oddělovač. Názvy akcí a zdrojů v Kiosku-/KConfigu tado omezení samozřejmě nemají a často využívají jiné znaky jako je například pomlčka a lomítko. Tyto znaky je potřeba buď odstranit, nebo nahradit. Vzniká tak reálná možnost kolize názvů.

Otázkou tedy je, zda má vůbec smysl pokračovat. Rozhodl jsem se, že implementuji alespoň to co půjde, abych lépe demonstroval nemožnost celé věci v testovací fázi.

3.2 Změny v KAuthu

Jak jsem již zmínil v sekci 3.1, je potřeba rozšířit KAuth tak, aby vracel vedle informace o úspěšnosti autorizace také zvláštní hodnotu pro případ, že akce není použitému systémem pro autorizaci známa. Důvodem je, že Všechny systémy, které KAuth podporuje umožňují toto zjistit a neměl by být tedy problém tuto vlastnost so KAuthu přidat.

3.2.1 PolicyKit a Policykit1

Tyto systémy jsou si velmi podobné. Ve výchozím stavu vypadá funkce pro získání autorizace takto: 3.1. Hodnota 'Unknown' je vrácena v případě, kdy dojde k chybě během autorizace. Důvod chyby lze zjistit bližším dotazováním autorizačního rozhraní. Uprava metody pak vypadá takto: 3.2. Při výsledku typu Unknown se zjistí, jestli byla chyba typu `E_CheckFailed` a v takovém případě se vrací hodnota `Action::Invalid`, která znamená, že jde o neznámou akci.

3.2.2 OSX Authorization Services

Tento framework umožňuje se přímo dotazovat na existenci akcí pomocí funkce `AuthorizationRightGet()`. Přidání podpory do KAuth je tak jednoduchou záležitostí přidání dotazu na existenci na začátek metody: 3.3.

```
Action::AuthStatus Polkit1Backend::actionStatus(const QString &action)
{
    PolkitQt1::UnixProcessSubject subject(QCoreApplication::applicationPid()
        );
    PolkitQt1::Authority::Result r =
        PolkitQt1::Authority::instance()->
        checkAuthorizationSync(action, &subject, PolkitQt1::Authority::None)
            ;
    switch (r) {
    case PolkitQt1::Authority::Yes:
        return Action::Authorized;
    case PolkitQt1::Authority::No:
    case PolkitQt1::Authority::Unknown:
        return Action::Denied;
    default:
        return Action::AuthRequired;
    }
}
```

Výpis 3.1: Autorizace akce v PolicyKit1

```
...
case PolkitQt1::Authority::Unknown:
    PolkitQt1::Authority::ErrorCode error =
        PolkitQt1::Authority::instance()->lastError();
    PolkitQt1::Authority::instance()->clearError()
    // E_CheckFailed should indicate that an action doesn't exist
    if(error == PolkitQt1::Authority::E_CheckFailed)
        return Action::Invalid;
    else // other errors. we treat them like before
        return Action::Denied;
...
```

Výpis 3.2: Autorizace akce v PolicyKit1 po úpravách

```
Action::AuthStatus AuthServicesBackend::actionStatus(const QString &action)
{
    // check if the action exists first, return error if not
    OSStatus exists = AuthorizationRightGet(action.toUtf8(), NULL);
    if(exists != errAuthorizationSuccess)
        return Action::Invalid;
...
```

Výpis 3.3: Ověření existence akce v OSX Authorization Services

3.3 Přesun vyhodnocení omezení na zdroje do KAuthorized

Zjišťování seznamu omezení na zdroje je umístěno v metodě `addCustomized()` třídy KStandardDirs a tento seznam je vytvářen před tím, než se načtou Kiosk profily. Tato metoda přidává profily k cestám pro načítání konfigurace a třída `KComponentData`, která tuto metodu volá následně způsobí znovunačtení konfigurace. Z takovéto načtené konfigurace ale již nejsou vytaženy seznamy omezení zdrojů.

Rozhodl jsem zjišťování seznamu omezených zdrojů přesunout do rozhraní `KAuthorized`, aby byly změny nutné pro integraci KAuthu pokud možno pouze na jednom místě.

Dále jsem se rozhodl, že oddělím zpracování tohoto seznamu od metody `addCustomized()` tak, aby se dalo spustit z třídy `KComponentData` poté, co `KStandardDirs` přidá Kiosk profily mezi konfiguraci. Docílí se tím toho, že je možné uložit omezení zdrojů stejným způsobem jako omezení na akce.

Implementoval jsem tedy funkci 3.4 pro získání seznamu omezení na zdroje ve formě objektu `QStringList`. Dále jsem přesunul vyhodnocení těchto omezení do nové metody `evaluateRestrictedResour()` v `KStandarDirs`, která je volána po znovunačtení konfigurace po přidání profilů. Bylo nutné také změnit konstruktor privátního objektu v KAuthorized, aby neblokoval zpracování profilů. Původně nebyl navržen na to, aby byl použit tak brzo během spouštění programů.

```
QStringList
KAuthorized::getRestrictedResourcesFromConfigGroup(const KConfig* config)
{
    QStringList result;

    // Process KIOSK restrictions.
    if (!kde_kiosk_admin || qgetenv("KDE_KIOSK_NO_RESTRICTIONS").isEmpty())
    {
        // probe KConfig-stored restrictions
        KConfigGroup cg(config, "KDE Resource Restrictions");
        const QMap<QString, QString> entries = cg.entryMap();
        for (QMap<QString, QString>::ConstIterator it2 = entries.begin();
        it2 != entries.end(); ++it2)
        {
            const QString key = it2.key();
            // resources are allowed by default (restricted = false here)
            if (!cg.readEntry(key, true))
            {
                result.append(key);
            }
        }
        // remove duplicates
        return result.toSet().toList();
    }
    // Kiosk restrictions don't apply, return empty list
    return result;
}
```

Výpis 3.4: Ukázka funkce přidané do KAuthorized

3.4 Specifikace omezení akcí a zdrojů

Ke specifikaci akcí v systému KAuth jsou využity statické .actions soubory. Nevýhody tohoto řešení jsem již uvedl, ale jinak to nejde. Jako základní jmenný prostor jsem se rozhodl použít `org.kde.kiosk`. Pro akce jsem se rozhodl pro `org.kde.kiosk.action` a pro zdroje `org.kde.kiosk.resource`.

Názvy akcí a zdrojů nemohou obsahovat jiné znaky než malá písmena a číslice. Je tedy nutné přeložit jejich skutečné názvy do formy, kterou je KAuth schopen použít. Z akce 'action/help' se tak stane 'actionhelp', společně se jmenným prostorem pak 'org.kde.kiosk.action.actionhelp'. Kdyby exitovala jiná akce s názvem 'action_help', došlo by ke kolizi.

3.5 Implementace konvertoru

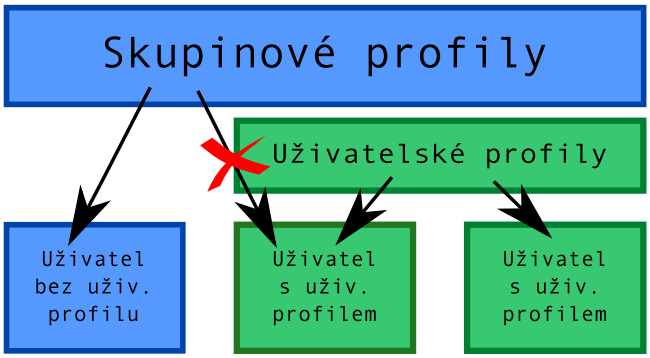
Konvertor se skládá ze dvou částí. Hlavní a nejdůležitější částí je pomocník, postavený na knihovně KAuth. Ten implementuje veškeré funkce celku (ve zdrojovém kódu je to kdelibs/kdecore/auth/kioskpklahelper.cpp). Druhou částí je jednoduchý terminálový program, který není až na spuštění pomocníka přes KAuth příliš zajímavý (ve zdrojových kódech je to kioskpklaconvert.cpp ve stejné složce jako pomocník).

Konvertor funguje tak, že nejdříve získá ze souboru /etc/kde4rc nastavení Kiosku, pomocí něj vyhledá Kiosk profily, komu a jakým skupinám jsou přiřazeny a jaké je pořadí skupin při zpracování skupinových profilů. Je také získán seznam akcí známých v PolicyKit1 Local Authority. Tento seznam se filtruje do dvou skupin podle jemnných prostorů použitých pro omezení na akce a zdroje. Toto jsou základní vstupní parametry programu. Uživatel nemá možnost do procesu přímo zasáhnout – program nemá žádné uživatelské vstupy.

Další částí je vytvoření .pkla souboru ze skupinových profilů. V pořadí zjištěném z nastavení Kiosku se z nich pomocí KConfigu načtou soubory globálního nastavení kdeglobals. Z těch jsou dále vytaženy skupiny KDE Action Restrictions a KDE Resource Retrictions. Klíče jejich položek jsou zpracovány stejně jako specifikace v .actions souborech. Upravené klíče společně s jejich hodnotami a přiřazenou skupinou pak tvoří záznamy ukládané do .pkla souboru pro skupiny. Soubor je uložen tak, aby byl zpracován dříve než soubory pro jednotlivé uživatele (jsou načítány v lexikografické pořadí podle názvu).

Dalším bodem je zpracování profilů přiřazených uživatelům. Postup je z části podobný jako u bodu předchozího, ale platí, že na uživatele s nastaveným uživatelským Kiosk profilem se skupinové profily nevztahují. Znamená to, že každý uživatel musí mít specifikovány všechny akce ve výchozím stavu (povoleno). K tomu se dále přidávájí omezení z uživatelského profilu. Ve výsledku se tak vyruší vliv skupinových profilů – pro ilustraci: 3.1. Pro každého uživatele je zvlášť vytvořen .pkla soubor.

Problémem řešení je zejména nemožnost reprezentace vlastnosti nezměnitelnosti z KConfigu. Omezení jsou jednou specifikována jako hodnota ano/ne v nastavení PolicyKit Local Authority a jsou tak z pohledu KConfigu nezměnitelná. Jedinou možností jak docílit toho, aby byla změnitelná je nezanést ji do PolicyKitu. Pak ale není možné s nimi v něm pracovat. Když je jednou akce jako 'action/help' pro ukázání menu s nápovědou zanesena do KAuthu/Policykitu, ztrácí nad ní uživatel kontrolu a nemůže tak menu schovat. To je poměrně závažný nedostatek.



Obrázek 3.1: Aplikace Kiosk profilů

3.6 Integrace KAuth do KAuthorized

K dokončení integrace zbývá už pouze jediný krok – volat z rozhraní KAuthorized metody systému KAuth. To se ukázalo být největším problémem ze všech. Vypadá to totiž, že KAuth zatím nikdy nebyl použit v takto velkém rozahu (autorizace všech akcí v KDE). Není tedy překvapením, že tato integrace odkryla některé problémy v KAuth a jím implementovaném PolicyKitu.

KAuth nabízí několik možností, jak autorizovat akci. V zásadě se jedná o různé metody třídy KAuth::Action, mající mírně odlišný význam. Prvním metodou je `execute()` – tato metoda je uvedena v příkladu [2] a je určena pro jednorázové blokující provedení akce. Pokud je to potřeba, uživatel je požádán o autentizaci. Metoda má podle [2] fungovat i bez přítomnosti pomocníka. `execute()` má také asynchronní verzi.

Další metodou je `authorize()`. Tato metoda je určena pro získání autorizace pro akce před tím, než by byla akce provedena. Má být používána ostatními metodami třídy `KAuth::Action`. Metoda podobně jako `execute()` může od uživatele vyžadovat autentizaci. Třetí metodou je `status()` – tato metoda je podobná `execute()`, ale v případech, kdy by byla vyžadována autentizace pouze o této skutečnosti informuje.

Takovýto význam by alespoň tyto metody měly mít. Skutečnost je ale odlišná. `KAuth::Action` používá pro autentizaci a použití pomocníků statické zásuvné moduly – to znamená, že jsou určeny při sestavení. Tyto moduly implementují lokálně dostupné autentizační mechanismy. Zde vzniká problém kvality těchto mechanismů a jejich implementace v KAuthu. To je také hlavním kamenem úrazu. Mnou používaný PolicyKit1 například vůbec neimplementuje metodu `authorize()`. Ta namísto výsledku vždy vrací hodnotu Action::Authorized. Její místo zastává metoda `status()`, která obaluje metodu `checkAuthorizationSync()` z knihoven polkit-qt-1. Toto je synchronní metoda pro autorizaci uživatele, která umožňuje upravit zda bude vyžadovat od uživatele autentizaci příznakem. Proč není implementována metoda `authorize()` je nejasné.

Ze všech uvedených metod je nejvhodnější, alespoň podle komentářů v kódu, metoda `status()`. Po konverzi z Kiosk profilů totiž nevznikají akce pro jejichž autorizaci by bylo nutné se autentizovat. Při použití této metody v kombinaci s PolicyKitem však dochází k nežádoucím jevům. Program, který používá takto upravené rozhraní KAuthorized totiž po čase 'zamrzne'. To stejné se stane i s celým sezením KDE. Zkoušel jsem použít i jiné metody, ale výsledky byly ještě horší.

Pro zjištění kde je chyba jsem vytvořil jednoduchý test kauthDoS. Ten v nekonečném

cyklu volá `KAuth::Action::status()` a v pravidelných intervalech vypisuje hlášení. Když program přestane vypisovat, znamené to, že došlo k chybě. Démon polkitd a program kauthDoS byly spuštěny v debuggeru gdb a byly získány "backtrace"z obou programů (výpisy 3.5 a 3.6). Podle těch to vypadá, že problém vzniká během komunikace jednotlivých částí PolicyKitu. Chybu jsem ohlásil jeho autorovi (David Zeuthen) a čekám na odpověď.

```
  #0  poll () from /lib/libc.so.6
2 #1  in socket_do_iteration () from /usr/lib/libdbus-1.so.3
  #2  in _dbus_transport_do_iteration () from /usr/lib/libdbus-1.so.3
4 #3  in _dbus_connection_do_iteration_unlocked () from /usr/lib/libdbus-1.so
    .3
  #4  in _dbus_connection_block_pending_call () from /usr/lib/libdbus-1.so.3
6 #5  in egg_dbus_connection_pending_call_block (
        connection=0x61a990, pending_call_id=196205)
8       at eggdbusconnection.c:2521
  ...
10 #16 in g_main_context_dispatch ()
        from /usr/lib/libglib-2.0.so.0
12 #17 in g_main_context_iterate ()
        from /usr/lib/libglib-2.0.so.0
14 #18 in g_main_loop_run ()
        from /usr/lib/libglib-2.0.so.0
16 #19 in main ()
```

Výpis 3.5: Backtrace z démona polkitd při 'zamrznutí' (zkrácený)

```
  #0  in poll () from /lib/libc.so.6
2 #1  in socket_do_iteration ()
    from /usr/lib/libdbus-1.so.3
4 #2  in _dbus_transport_do_iteration ()
    from /usr/lib/libdbus-1.so.3
6 #5  in egg_dbus_connection_pending_call_block (
    connection=0x6add50, pending_call_id=74401)
8     at eggdbusconnection.c:2521
  #6  in polkit_authority_check_authorization_sync ()
10    from /usr/lib/libpolkit-gobject-1.so.0
  #7  in PolkitQt1::Authority::checkAuthorizationSync ()
12    from /home/kde-devel/kde/lib/libpolkit-qt-core-1.so.0
  #8  in KAuth::Polkit1Backend::actionStatus ()
14    at kdelibs/kdecore/auth/backends/polkit-1/Polkit1Backend.cpp:87
  #9  0x000000000040162e in main (argc=1, argv=<value optimized out>)
16    at kdelibs/kdecore/auth/kauthDoS.cpp:40
```

Výpis 3.6: Backtrace z testovacího polkitd při 'zamrznutí' (zkrácený)

Systém KAuth za využití PolicyKit1 tedy do jisté míry funguje, není ale nijak spolehlivý. Co funguje dobře jsou pomocníci. Ti mají zpravidla několik málo akcí, které se používají pouze pokud je to nutné – příkladem budiž KCM modul pro nastavení času, nebo mnou implementovaný konvertor pro Kiosk profily. Při integraci do rozhraní KAuthorized, které je využíváno téměř každým programem v KDE je však objem dotazů o několik řádů vyšší. Sezení KDE tak přestane reagovat již během spuštění. Chyba v podstatě umožňuje přihlášenému uživateli provést DoS[1] útok na PolicyKit, protože polkitd běží jako systémová

[1] Denial of Service

služba, společná pro všechny uživatele. V dalším testování tedy bohužel nelze smysluplně pokračovat, dokud nebude tato chyba odstraněna.

3.7 Návrh dalšího postupu

Nalezené nedostatky jsem již myslím popsal dostatečně. Je tedy na místě navrhnout, jak by se daly řešit. V prvé řadě je potřeba opravit zmíněnou chybu v PolicyKitu.

Dále je potřeba rozšířit KAuth:

1. Je potřeba, aby uměl hlásit, pokud autorizovaná akce není známá – a to pro všechny podporované autorizační systémy. Toto jsem udělal a otestoval pro metodu `status()` a PolicyKit1.

2. KAuth musí umět nejen ověřovat a spouštět akce, ale také měnit oprávnění k těmto akcím pro uživatele a skupiny. Zde je zatím k dispozici pouze KCM modul pro PolicyKit v balíku policykit-kde-1, ale ten je zaměřen pouze na uživatele. Zcela chybí možnost nastavit oprávnění z kódu programu.

3. Mělo by také být možné přidávat nové akce. To znamená, že by se nemusely do systému instalovat statické soubory s definicemi akcí.

Body 2 a 3 by bylo možné realizovat jakožto pomocníky a přidat do systému KAuth rozhraní, které by umožnilo s nimi pracovat.

Korektní integraci KAuthu přímo do KAuthorized považuji kvůli omezením ze strany KAuthu za nemožnou. Oba systémy jsou navíc velmi rozdílné v některých kritických bodech, i když to tak na první pohled nevypadá. I kdyby byla integrace úspěšná, nezískalo by se tím téměř nic. Na pouhé dotazování se na to, jestli je uživateli dovoleno provádět nějakou akci není potřeba KAuth. Stačí jakákoliv databáze. To však nebrání využití KAuthu převážně tam, kde použití KAuth pomocníků poskytuje větší bezpečnost – izolují se části programu vyžadující administrátorská práva od zbytku kódu. Z pohledu administrátora je možnost udělit konkrétním uživatelům a skupinám oprávnění tyto pomocníky spouštět snad největší výhodou řešení.

Dalším možným rozšířením by bylo nějakým způsobem spojit KAuth se systémy jako je SELinux[2]. Zde by byla výhoda v tom, že akce, které by jinak byly uživateli umožněny jde omezit na úrovni operačního systému. Využitelnost KAuthu by se tak rozšířila například na omezení spouštění programů a použití knihoven (příkladem budiž KCM moduly) a skutečné omezení zdrojů dat. Například pokud by měl uživatel zakázáno používat vlastní pozadí na plochu, bylo by zamezeno programu plasma-desktop načítat jiné obrázky, než ty instalované do systému administrátorem - a to bez možnosti to obejít. Omezení na zdroje tak jak je v KDE4 je v porovnání s takovým řešením v podstatě bezzubé. Potom by mělo smysl odstranit některé typy omezení ze systému KConfig, a přímo je integrovat do KAuth. Za jiných podmínek to však považuji za nesmysl.

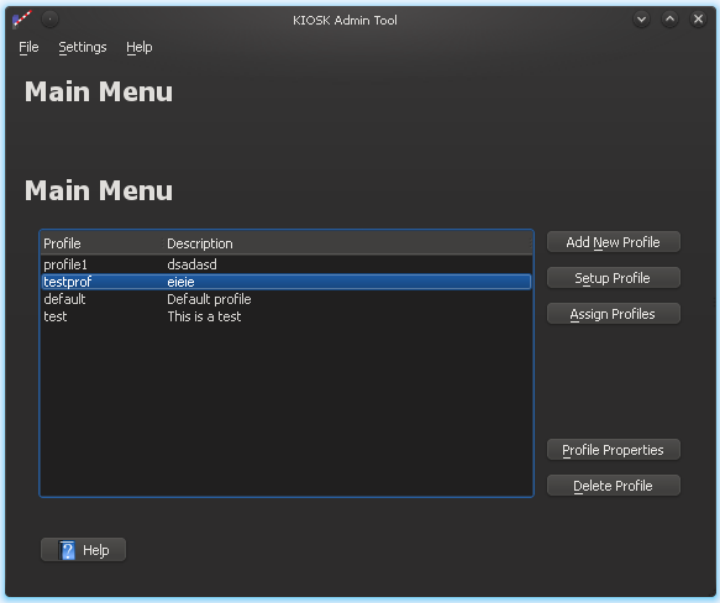
Co se týče kombinace Kiosk, KAuthorized a KConfig, je rozhodně co dohánět. Bylo by dobré tyto části KDE pročistit a zdokumentovat. Zdvojení funkcí `KAuthorized::authorize()` a `KAuthorized::authorizeKAction()` je poněkud chaotické a vede ke zmatkům, kdy je jeden název akce používán s funkcemi náhodně. `KAuthorized::authorizeKAction()` by měla přijímat jako parametr pouze reference typu KAction, aby se takovým zmatkům předešlo.

[2]Security Enhanced Linux

Kapitola 4

Nástroj KioskTool

Starší verzi nástroje pro KDE3 jsem již popsal. Tato kapitola bude tedy věnována portu na KDE4 a jakým způsobem funguje. Navrhnu pro něj nové uživatelské rozhraní a opravím chyby, aby bylo možné program začít používat.



Obrázek 4.1: Uživatelské rozhraní se od verze z KDE3 příliš nezměnilo

Port nástroje do KDE4 totiž již existuje, i když je nedokončený a dá se říci opuštěný. Uživatelské rozhraní se příliš nezměnilo: 4.1. Původní nastavení pomocí velkého XML souboru bylo odstraněno a nahrazeno mnoha menšími .ini soubory (samozřejmě používají KConfig). To má za účel umožnit ostatím autorům napsat si pro své aplikace rozšíření. Nástroj však ztratil většinu svých starých vlastností, schopnost spouštět KControl moduly a náhledy, své pěkné (i když zbytečné) obrázkové vzezření a získal několik dalších chyb.

Strukturální popis Aplikace KioskTool se skládá z několika základních komponent a využívá grafické rozhraní navržené pomocí nástroj Qt Designer (části rozhraní jsou specifikovány v .ui souborech, ze kterých se při kompilaci generuje kód). Vzhled je tedy alespoň

z pohledu programátora částečně oddělen od funkce programu. Grafické prvky programu však přímo obsahují data se kterými se pracuje – není využito návrhového vzoru MVC[1].

Při startu programu jsou nejdříve vytvořeny základní komponenty (`KAboutData`, `KApplication`) a hlavní komponenta grafického rozhraní `KioskGui`. Ta je zobrazena. Potom je nastartováno vyhodnocování událostí.

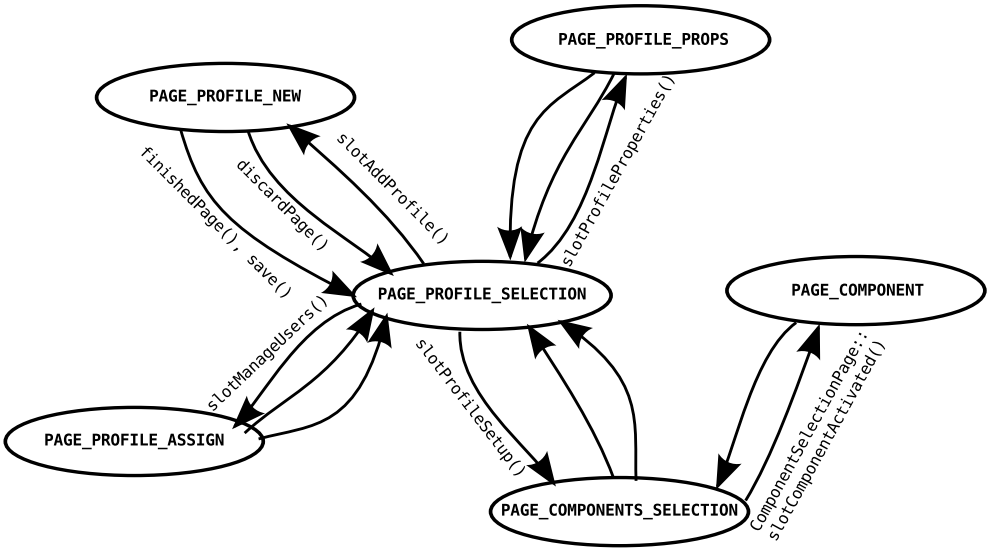
Komponenta `KioskGui` je odvozena od třídy `KXmlGuiWindow` – načítá tedy část svého vzhledu z .rc souboru 4.1 ve formátu XML.

```
<?xml version="1.0"?>
<!DOCTYPE gui SYSTEM "kpartgui.dtd">
<gui name="kioskgui" version="3.1">
<MenuBar>
 <Menu name="file">
    <Action name="upload_all"/>
 </Menu>
</MenuBar>
</gui>
```

Výpis 4.1: kiosktoolui.rc

`KioskGui` potom vytváří instance tříd `KioskRun` a `MainView`. `KioskRun` je pouhou obálkou nad souborem funkcí různého určení (tzv. God object anti-pattern) a je používána pro většinu manipulace s profily a spouštění dalších programů. `MainView` pak určuje vzhled celého programu. Je to třída generovaná z .ui souboru, obsahuje dvě úrovné nadpisů, tři tlačítka a objekt typu `QStackedWidget`, který je určen pro zobrazení aktuální stránky. Při startu je to stránka se seznamem profilů (`PAGE_PROFILE_SELECTION`).

Přechod mezi stránkami je prováděn pomocí metody `selectPage(enum page)`, volané v reakci na akce uživatele a lze v hrubých obrysech popsat pomocí stavového automatu 4.2.



Obrázek 4.2: Stavy grafického rozhraní aplikace KioskTool

Ve stavech `PAGE_PROFILE_NEW` a `PAGE_PROFILE_PROPS` je používán stejný typ stránky. Jednou

[1] Model-View-Controller

pro vytvoření nového profilu, podruhé pro změny v něm. `PAGE_PROFILE_ASSIGN` je stav, ve kterém je aktivní stránka pro přiřazení profilů uživatelům a skupinám. Ve stavu `PAGE_COMPONENTS_SELECTION` je načten profil a zobrazena je stránka se seznamem komponent pro stav `PAGE_COMPONENT` – zde přestává stačit jeden stavový diagram.

Je zřejmé, že způsob jakým je vytvořeno grafické rozhraní přímo určuje funkci programu. Přechod na stránku se seznamem komponent zapříčiní otevření profilu. Návrat na hlavní stránku se seznamem profilů pak způsobí jeho uložení. Platí to i naopak - technická omezení kladená na některé funkce programu se odrážejí v návrhu jeho grafického rozhraní. Například program nemůže upravovat více jak jeden profil, protože ve třídě KioskRun nastavuje pro spuštění KConfig modulů proměnné prostředí a kopíruje profil do dočasného umístění, kde ho může případná spouštěná aplikace měnit. To by se mělo dát rozšířit – program by měl např. být schopen pracovat s profily na pozadí, zobrazovat okno pro úpravy konkrétního profilu (ať už vlastní nebo z jiného programu) a zároveň třeba i druhé okno se seznamem všech uživatelů a k nim přiřazených profilů. Je tedy potřeba oddělit model pro práci s Kiosk profily a jeho stavy od kódu pro grafické rozhraní.

Vzniká tak také nutnost práce s asynchronními událostmi, protože provedení některých funkcí modelu může trvat déle než je přípustné a blokovat tak uživatelské rozhraní. V aplikaci, kde se pouze přechází mezi přesně vymezenými stavy je toto omluvitelné, ale pokud má uživatel otevřeno několik oken, nebude chtít například čekat, až se profil uloží na server.

Po těchto úpravách bude možné vylepšit uživatelské rozhraní programu. Stanovím si tedy několik cílů:

1. Krátkodobě - otestovat program a opravit v něm chyby, tuto verzi zveřejnit (bude součástí práce).
2. Navrhnout a implementovat model pro práci s profily. Měl by být bezpečný a pro přístup k profilům používat KAuth pomocníka. Nabízí se použít Qt API pro stavové automaty [4].
3. Musí být možné upravovat profily zároveň z více aplikací, případně ručně. KioskTool toto nijak neřeší a je možné ztratit data pokud je spuštěn dvakrát.
4. Implementovat několik jednoduchých terminálových programů pro práci s profily.
5. Nakonec vytvořit nové grafické rozhraní pro KioskTool.

Triviální chyba Při testování jsem narazil pouze na jednu drobnou, ale závažnou chybu. Při přidělování profilů není možné přiřadit více jak jeden profil pro uživatele nebo skupiny. To znamená, že program není vůbec použitelný ke svému účelu. Problém má triviální řešení. Přiřazení se totiž řídilo tímto algoritmem:

```
  for( int idx = 0; idx < listGroups->topLevelItemCount(); ++idx )
2 {
      item = listGroups->topLevelItem(idx);
4     if (item->text(0) == group)
          break;
6 }
  if (item)
8 {
      Error();
```

To je zjevně špatné, protože proměnná item bude obsahovat nenulovou hodnotu, jakmile pude přiřazen alespoň jeden profil. Je tedy potřeba přidat za `break;` další řádek s `item = 0;`.

4.1 Návrh uživatelského rozhraní

Kapitola 5

Závěr

V kapitole 2 byly popsány technologie a rozhraní. V kapitole 3 byl učiněn pokus o integraci KAuth so KAuthorized. Byly přitom odkryty závažné nedostatky v KAuth a rozhraních, nad kterými funguje. Nejzajímavější je pak Denial of Service útok na PolicyKit, který jsem objevil za pomocí implementovaného testovacího nástroje kauthDoS. KAuth byl integrován do KAuthorized, al díky zmíněné chybě je toto naprosto nepoužitelné. Dále byl implementován nástroj kioskpklaconvert a KAuth pomocník kioskpklahelper. Ty umožňují konvertovat omezení akcí a zdrojů z KConfigu do nastavení PolicyKit Loacl Authority. Kapitola 4 popisuje port nástroje KioskTool do KDE4 a navrhuje pro něj nové grafické rozhraní a s tím spojené hloubkové změny. K implementaci jsem se zde nedostal, ale opravil jsem v nástroji chyby (alespoň ty, které jsem našel).

Literatura

[1] Bobčík, B.: PAM - správa autentizačních mechanismů [online]. http://www.root.cz/clanky/pam-sprava-autentizacnich-mechanismu/, 2000-09-19 [cit. 2010-05-16].

[2] Freddi, D.: Using KAuth actions in your application [online]. http://techbase.kde.org/Development/Tutorials/KAuth/KAuth_Actions, 2010-02-12 [cit. 2010-05-16].

[3] Komunita KDE: Kiosk/Introduction [online]. http://techbase.kde.org/KDE_System_Administration/Kiosk/Introduction, 2008-03-14 [cit. 2010-05-16].

[4] Nokia Corporation: The State Machine Framework [online]. http://doc.qt.nokia.com/4.6/statemachine-api.html, 2010 [cit. 2010-05-16].

[5] Zeuthen, D.: Referenční manuál k PolicyKitu [online]. http://hal.freedesktop.org/docs/polkit/, 2009-07-24 [cit. 2010-05-16].

[6] Zeuthen, D.: Manuálová stránka PolicyKit1 Local Authority [online]. http://hal.freedesktop.org/docs/polkit/pklocalauthority.8.html, 2010 [cit. 2010-05-16].

[7] Zeuthen, D.: Manuálová stránka PolicyKit1 [online]. http://hal.freedesktop.org/docs/polkit/polkit.8.html, 2010 [cit. 2010-05-16].