Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE

Martin Kudlvasr

Příprava VR prezentací z CAD modelů

Kabinet software a výuky informatiky
Vedoucí diplomové práce: Prof.Ing. Jiří Žára, CSc.

Studijní program: Informatika, Softwarové systémy

Chtěl bych poděkovat Prof. Ing. Jiřímu Žárovi, CSc. za vedení diplomové práce a za to, že mi umožnil získat ve společnosti ŠKODA AUTO, a. s. spoustu nových zkušeností, Ing. Leoši Červenému za obětavou podporu při komunikaci s ostatními pracovišti a Ph.D. Antonínu Míškovi za bezprostřední pomoc při řešení problémů přímo na pracovišti virtuální techniky.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 16.4.2008

Martin Kudlvasr

Obsah

Název práce: Příprava VR prezentací z CAD modelů

Autor: Martin Kudlvasr

Katedra (ústav): Kabinet software a výuky informatiky, MFF UK

Vedoucí diplomové práce: Prof.Ing. Jiří Žára, CSc.

e-mail vedoucího: zara@fel.cvut.cz

Abstrakt: Analýza specifického problému pracoviště virtuální techniky ve společnosti ŠKODA AUTO, a. s., návrh řešení a implementace. Zaměstnanci při přípravě virtuálních scén (prezentací nových typů automobilů) používají program Virtual Designer 2, který selhává při správě verzí prezentací. Udržování verzí je pro zaměstnance zdlouhavé a je náchylné k chybám. Program VR Aid tento problém řeší a snaží se o maximální zjednodušení často se opakujících úkonů. Disponuje grafickým uživatelským rozhraním s intuitivním ovládáním. Program umožňuje zrychlit přípravu nových verzí virtuálních prezentací a šetří tak cenný čas zaměstnanců. Byl navržen tak, aby doplňoval a zároveň nenarušoval funkce stávajícího programového vybavení pracoviště, spolupracuje zejména s aplikacemi Virtual Designer 2, Opera a DeltaGen. Součástí práce je nově vytvořená dokumentace používaných formátů FHS a VRP a knihovny pro práci s těmito formáty. Při práci byl vyvinut (univerzálně použitelný) systém pro zjednodušení komunikace mezi vývojářem a zákazníkem.

Klíčová slova: virtuální realita, FHS, Virtual Designer 2, ŠKODA AUTO, a. s., Python

Title: Preparation of VR presentations from CAD models

Author: Martin Kudlvasr

Department: Dept. of Software and Computer Science Education, Faculty of Mathematics and Physics, Charles University in Prague

Supervisor: Prof. Ing. Jiří Žára, CSc.

Supervisor's e-mail address: zara@fel.cvut.cz

Abstract: Analysis of a specific problem in virtual engineering department in ŠKODA AUTO, a. s. company, concept of the problem solution and implementation of the solution. Virtual engineering department employees use a Virtual Designer 2 application for creation of virtual reality worlds (new car concept presentations). This application does not fit well for the task of creating versions of these presentations. Maintaining versions using this program is a lengthy and fault-prone process. Application VR Aid solves this problem. It reduces and simplifies frequently repeating tasks and uses intuitive graphical user interface. VR Aid allows to speed up the process of creating new versions of presentations and spares precious employee time. It was designed to complement (and not to disrupt) currently used computer applications (cooperates with Virtual Designer 2, Opera and DeltaGen). Results of this work include new documentation of FHS and VRP file formats. During the application development a new system of communication between customer and developer was created. This system of communication is generaly usable for GUI application deveopment.

Keywords: virtual reality, FHS, Virtual Designer 2, ŠKODA AUTO, a. s., Python

1 Virtuální realita ve ŠKODĚ AUTO, a. s.

Návrh automobilů ve ŠKODĚ AUTO, a. s. je dnes již z velké většiny digitalizován. Od návrhu jednotlivých dílů přes design, prostorové řešení interiéru a konstrukci nástrojů až po deformační testy dílů a bourací testy celých automobilů.

Začíná se od návrhu jednotlivých dílů. Pro tento účel se ve ŠKODĚ AUTO, a. s. nejčastěji používá program CATIA (vyvíjený francouzskou firmou Dassault Systemes a prodávaný IBM). Jako výměnný formát mezi aplikacemi slouží standard IGES (ANSI standard pro CAD data).

Ať už je ale konstrukční program jakýkoli, všechna dokumentace se shromažďuje v databázi nazývané KVS [ká-fau-es]. Tato databáze obsahuje úplně všechno, co kdy bylo ve společnosti Volkswagen vykonstruováno a zdokumentováno. Zaměstnanci k této databázi přistupují buď online (http klientem) nebo též programem HyperMona, který usnadňuje vyhledávání a stahování dat z KVS.

Pro co nejvyšší urychlení vývoje je důležité, aby změny na automobilu, které udělá oddělení konstrukce, mohly být zodpovědně posouzeny co nejdříve. Zodpovědně posoudit změny mohou pouze vyšší vedoucí ŠKODA AUTO, a. s. Při posuzování změny zodpovědným vedoucím je potřeba, aby se jeho vnímání vykonstruované části lišilo co možná nejméně od skutečnosti (od skutečně vyrobeného automobilu/části). Proto se používají metody virtuální reality, které umožní rychle posoudit nové varianty a tím zrychlit vývoj.

Pro trojrozměrné zobrazování se používá virtuální stěna nebo CAVE. CAVE je tvořen několika promítacími plochami (obvykle 3-4 stěny, a podlaha) a spolu s brýlemi vytváří mimořádně dokonalý vjem trojrozměrného prostoru a pocit obklopení.

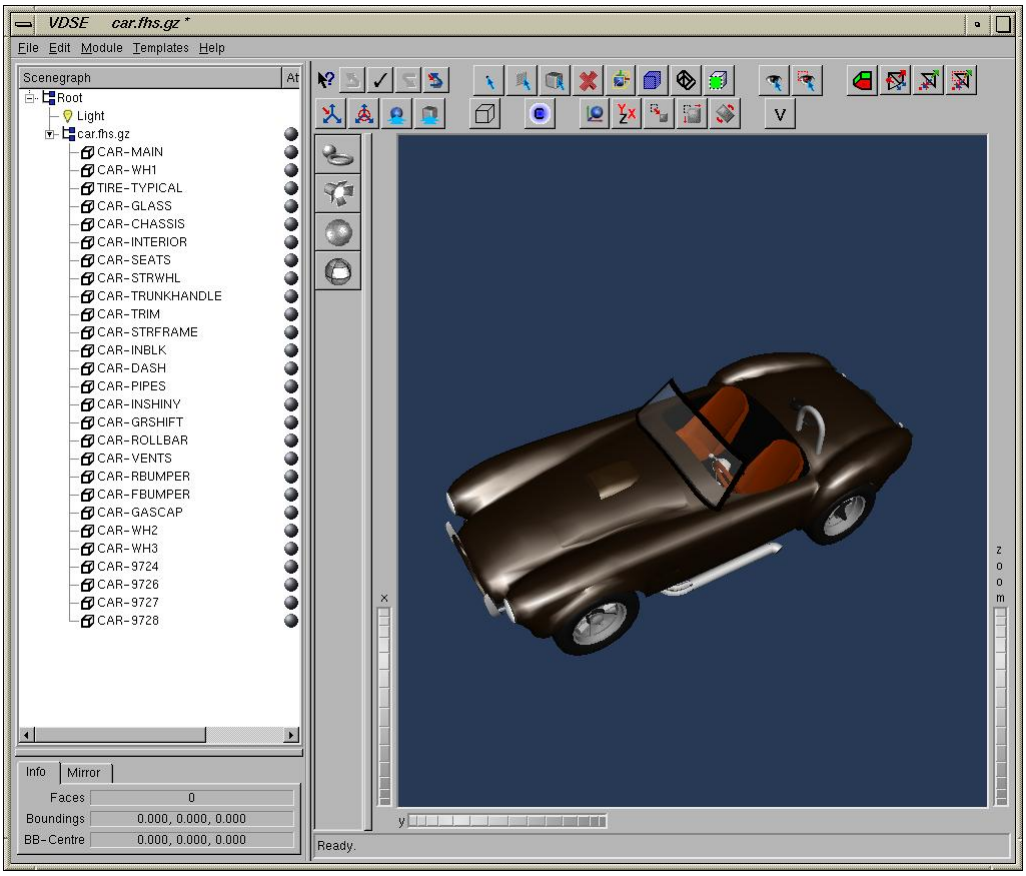
Obrázek 1: systém pro virtuální realitu CAVE

1.1 Pracoviště virtuální techniky

Prezentace pro odpovědné vedoucí je nutno z narýsovaných dílů nejprve vytvořit. To je úkolem pracovišti virtuální techniky. Zde se musejí zaměstnanci vybrat z KVS potřebné díly, překonvertovat je do vhodného formátu, vytvořit scénu, přiřadit k dílům vhodné materiály pro zobrazení (matné, lesklé, barevné) a případně vytvořit skript pro drobnou animaci (otevírání dveří, změna konfigurace sedadel). Členům tohoto pracoviště se ve ŠKODĚ AUTO, a. s. říká VR-mani a budou tak nazýváni i v této práci.

1.2 Virtual Designer 2

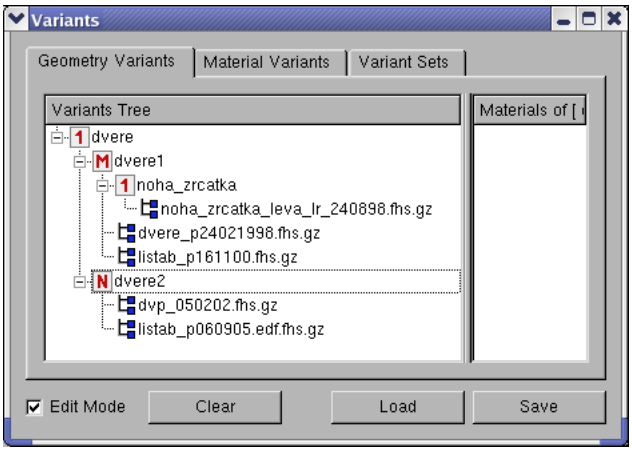
Zkráceně VD2. VR-mani ho používají jako svůj hlavní nástroj. VD2 slouží pro konstrukci scény, správu stromu scény, přiřazování materiálů a vytváření animací. K ukládání scény používá VD2 svůj vlastní formát FHS.



Obrázek 2: Ukázka práce v programu VD2

1.3 VRP

Při přípravě prezentace je vhodné vytvořit rovnou více variant. Při prezentaci ve virtuálním studiu pak může VR-man rychle změnit zobrazení prezentace a tak zodpovědnému vedoucímu předvést několik variant návrhu. Přidruženou funkcí VD2 je spravovat tyto varianty. Přidruženou proto, že varianty se neukládají ve formátu FHS, ale ve formátu VRP. Prezentace je tak reprezentována na disku dvěma soubory, přičemž ze souboru VRP vedou odkazy na části souboru FHS.



Obrázek 3: Ukázka práce s variantami v programu VD2

Opera

Formát FHS nepracuje s obecnými plochami ani s křivkami, ale pouze s trojúhleníky. Na druhou stranu CAD modely ve formátech CATIA nebo IGES jsou typicky tvořeny právě obecnými křivkami a plochami. CAD modely je tedy třeba před vytvářením prezentace překonvertovat do formátu FHS.

Konverzi formátů CATIA nebo IGES do FHS provádí VR-man pomocí programu jménem Opera [2] .V CATIA a IGES jsou data uložena jako obecné křivky. Jak bylo uvedeno výše, formát FHS pracuje pouze s trojúhelníky. Je proto třeba při konverzi specifikovat maximální odchylku od ideální křivky (jak jemné trojúhelníky mají být). Čím menší odchylka, tím menší trojúhelníčky. Této maximální povolené odchylce se říká *tessellace*.

V době vypsání této práce běžela Opera pouze na platformě IRIX, dnes již běží nativně i na platformě Windows. Kromě Opery je možné pro konverzi použít i program DeltaGen. Ten je schopen například navazovat generované pláty trojúhelníčků v bodech společných oběma plátům. Opera toto neumí a tak může dojít uprostřed zdánlivě celistvé plochy k prosvítání pozadí.

1.4 DeltaGen

Koncem roku 2007 začalo pracoviště virtuální techniky přecházet na program DeltaGen. Firma RTT AG, která DeltaGen vyvíjí již před časem koupila firmu vrcom GmbH a přestala VD2 vyvíjet. Program DeltaGen používá binární formát CSB a nabízí mnohem pokročilejší funkce při modelování než VD2. Na druhou stranu DeltaGenu mnoho funkcionality chybí - například modelování srážek, které umí zase VD2.

2 Analýza problému

2.1 Analýza práce VR-mana

Před začátkem práce na programu jsem dlouho zjišťoval, jaké úkony VR-man v průběhu života prezentace. Nejedná se tady jen o vytvoření, ale i o pozdější udržování prezentace v aktuálním stavu.

Definice: Prezentace

Prezentace je celý soubor FHS, se kterým VR-man pracuje a který bude později předvádět ve virtuálním studiu. Jednotlivé díly čerstvě překonvertované pomocí Opery tedy prezentací nejsou, i kdyz jsou také uloženy ve formátu FHS. Prezentace může (ale nemusí) obsahovat soubor variant VRP.

2.1.1 Postup při vytváření prezentace

V tabulce jsem shrnul postup VR-mana, jak je náročné krok provádět a v jakém prostředí (resp. s jakým nástrojem) přitom VR-man pracuje.

stažení dílů z KVS	lze provést hromadně	HyperKVS nebo HyperMona
přeložení dílů do formátu FHS	lze provést hromadně	Opera
vložení dílů do kostry prezentace	ručně	VD2
přiřazení materiálů	ručně	VD2
vytvoření variant	ručně	VD2

Definice: Díl

Díl je CAD soubor (nebo FHS po zkonvertování), který je nebo byl samostatně uložen v databázi KVS.

Definice: Kostra prezentace

Kostra prezentace je prezentace, do které VR-man ještě nevložil žádné díly. VR-man má typicky díly v prezentaci rozděleny do několika skupin, aby se mu s díly lépe pracovalo. Názvy a rozložení těchto skupin se často opakuje a VR-man má často takovouto prázdnou kostru prezentace uschovanou na svém disku.

HyperKVS a HyperMona jsou uživatelské nástroje pro práci s databází KVS. HyperKVS je možno používat pomocí webového prohlížeče, HyperMona je samostatný program.

Při stahování dílů z KVS jsou tyto díly pojmenovány podle toho, jaké parametry měly v KVS. Jde zejména o ID dílu, verzi dílu, datum vytvoření této verze, krátký textový popisek/pojmenování. Stejně jako soubor je následně pojmenován uzel v FHS. VR-man samozřejmě může toho chování HyperKVS/HyperMony a VD2 změnit. Obvykle k tomu ale nemá důvod. Velká většina dílů v prezentaci je takto pojmenována. Z prezentace by tedy bylo možné nějakým programem tyto parametry dílů získat. Stejně tak je obvykle možné získat

parametry dílu ze jména souboru na disku.

Definice: ID dílu

Jednoznačný identifikátor dílu používaný v KVS. Skládá se z 9 alfanumerických znaků ve třech skupinách (XXX.XXX.XXX). První skupina znaků určuje typ automobilu. Poslední skupina dílů určuje díl v rámci automobilu a je často stejná pro několik typů (např. zadní nárazník bude mít stejné poslední 3 znaky u několika typů automobilů).

Při překladu dílu programem Opera dochází k tessellaci (trojúhelníčkování) dílu. VR-man obvykle používá Operu tak, že vstupní a výstupní soubor se jmenují stejně a liší se pouze koncovkou. Ze jména přeloženého souboru tedy není možné získat informaci o tom, s jakou tessellací byl překládán.

Celý postup při vytváření prezentace je tvůrčí, tj. neobsahuje monotónní a opakující se úkony. V této fázi není důvod postup VR-mana měnit.

2.1.2 Postup při přidávání nových dílů

Byly vykonstruovány nové verze dílů a VR-man dostal za úkol zapracovat je do prezentace.

stažení dílů z KVS	lze provést hromadně	HyperKVS nebo HyperMona
přeložení dílů do formátu FHS	lze provést hromadně	Opera
mazání starých dílů a vkládání nových na jejich místo	nutno provádět po jednom ručně	VD2
úprava nových dílů tak, aby odpovídaly úpravám provedených na starých dílech (zejména zrcadlení a nastavení materiálu)	nutno provádět po jednom ručně	VD2

Opakující se činnost při vkládání dílů by bylo možné zjednodušit nějakým programem. Z FHS souboru prezentace je možné získat informaci o parametrech dílu a ze souborů na disku také (název dílu v FHS byl dříve jménem souboru na disku a ten vznikl stažením z KVS). Po získání těchto parametrů by bylo možné porovnat verze dílů a v prezentaci použít díl s vyšší verzí.

2.1.3 Postup při přidávání nových dílů spolu s vytvořením další verze prezentace.

Byly vykonstruovány nové verze dílů a VR-man dostal za úkol zapracovat je prezentace. Zároveň musí VR-man zachovat dosavadní podobu prezentace (ve stejném souboru), aby byl při prezentaci ve virtuálním studiu schopen ukázat starou i novou verzi.

Postup VR-mana:

stažení dílů z KVS	lze provést hromadně	HyperKVS nebo HyperMona
přeložení dílů do formátu FHS	lze provést hromadně	Opera
vytvoření prostoru v prezentaci pro novou verzi	ručně	VD2

zkopírování těch částí prezentace, kde se změnily díly	ručně	VD2
v prostoru pro novou verzi - mazání starých dílů a vkládání nových na jejich místo	nutno provádět po jednom ručně	VD2
v prostoru pro novou verzi - úprava nových dílů tak, aby odpovídaly úpravám provedených na starých dílech (zejména zrcadlení a nastavení materiálu)	nutno provádět po jednom ručně	VD2
znovuvyvoření variant pro novou verzi	ručně	VD2

Vytvoření prostoru pro novou verzi a kopírování částí prezentace by určitě šlo automatizovat. VR-man kopíruje pouze ty skupiny dílů původní prezentace, ve kterých se nějaký díl změnil. Není to jen z důvodu šetření diskovým prostorem. Při práci s velmi velkými soubory (přes 1GB) je již potřeba počítat i s časem načítání souboru z disku do operační paměti. Zjišťování, ve které části prezentace se nějaký díl změnil je pro VR-mana zdlouhavé. Přitom program, který by dokázal získat parametry dílů z prezetace a z dílů na disku by tento problém dokázal jednoduše vyřešit.

Nejvíce frustrujícím krokem je pro VR-mana znovuvytváření všech variant. Ve VD2 je možné zkopírovat najednou část FHS stromu. VD2 automaticky vytvoří nová UID. Pokud však prezentace obsahovala varianty, pak tyto varianty budou i nadále ukazovat na stará UID. Udělat "sdružené" kopírování v FHS i VRP najednou, které by zachovávalo odkazy možné není. VR-man tak musí znovu vykonat všechny kroky, které prováděl při vytváření prezentace. Tyto kroky samozřejmě nemá nikde zaznamenány a tak snadno dojít k odlišnostem.

Definice: UID

UID je šestnáctimístné hexadecimální číslo identifikující jeden FHS uzel v prezentaci. UID mají zejména uzly, které obsahují geometrická data (tag GEOMETRY), a uzly, které seskupují ostatní uzly (mají tag ASSEMBLY). Některé uzly UID nemají (například DEFMAT - definice materiálu).

Přitom jednoduchý program by situaci vyřešil (zkopírovat část FHS stromu=> vygenerovat nová UID => zkopírovat část VRP stromu => zaměnit odkazovaná UID tak, aby odpovídala kopii v FHS).

2.1.4 Postup při změně tessellace

VR-man má za úkol pozměnit prezentaci tak, aby bylo možné ukázat větší detaily. Je třeba a by detaily byly hladké a nebyly vidět žádné hrany tam, kde má být zaoblená plocha.

shromáždění všech dílů od všech verzí	lze provést hromadně/obvykle jsou již na lokálním disku	HyperKVS nebo HyperMona
přeložení dílů do formátu FHS s novou tessellací	lze provést hromadně	Opera
mazání starých dílů a vkládání nových na jejich místo	nutno provádět po jednom ručně	VD2
úprava nových dílů tak, aby odpovídaly úpravám provedených na starých dílech (zejména zrcadlení a nastavení materiálu)	nutno provádět po jednom ručně	VD2

Definice: tessellace

Proces převodu plochy (nebo více ploch) z křivkové reprezentace do ploškové.

Definice: velikost tessellace

Velikost tessellace je číslo udávající maximální vzdálenost od ideálního povrchu. Při tessellaci na něm závisí jemnost výsledných plošek. Čím větší tessellace, tím větší povolená odchylka, tím větší plošky a tím méně dat.

VR-man volí velikost tessellace vždy co největší, jaká je přípustná pro daný typ prezentace ve virtuálním studiu. Se zmenšováním tessellace prudce roste objem dat, se kterými se následně obtížně pracuje. Změna tessellace v celé prezentaci je proto docela častým úkonem. Pro VR-mana to ovšem znamené spoustu monotónní práce, kdy musí každý jednotlivý díl znovu načíst z disku a nahradit jím ten starý (a upravit ho stejně jako ten starý - materiál, zrcadlení). To představuje velké množství mravenčí práce. Často se stane, že VR-man na nějaký díl zapomene, změna se neprojeví na jeho pracovní stanici, ale až ve virtuálním studiu.

Jak by vypadal program, který by měl tuto situaci řešit? Musel by nějak získat informaci o tessellaci dílů v prezentaci a o tessellaci dílů na disku. Informaci o tessellaci dílů v prezentaci by šlo získat jen velmi obtížně (díl se tam mohl ocitnout jakkoli). Bylo by však možné získat informaci o dílech na disku - například zhromažďovat je do adresářů podle toho, s jakou tessellací byly přeloženy. Pak by mohl program jednoduše vyměnit všechny díly v prezentaci, ke kterým by na disku existovaly díly v potřebné tessellaci. Nějaká práce programu by byla sice zbytečná (někdy by vyměňoval díl za naprosto totožný), VR-man však nemusí dělat téměř nic.

2.2 Spolupráce VRAid se stávajícími programy

Aplikace VR Aid by měla spolupracovat se stávajícím programový vybavením pracoviště virtuální techniky. Tím je zejména program VD2.

2.2.1 Vytváření nové prezentace

Když VR-mani vytváření novou prezentaci pomocí programu VD2, neobsahuje tato činnost žádnou zbytečnou nebo často se opakující aktivitu. Program VR Aid tuto funkcionalitu znovu neimplementuje. Těžko by dosáhl stejného stupně použitelnosti při samotném vytváření prezentací.

2.2.2 Vytváření dalších verzí prezentace

Vytváření dalších verzí prezentace je pro VR-mana práce zdlouhavá a plná opakujících se kroků. V tomto případě je vhodné vytvořit novou funkcionalitu a zjednodušit tak práci VR-mana.

Je třeba, aby při automatické úpravě prezentace nebyla zmařena práce, která byla již dříve do prezentace vložena.

Vytváření animací

Při vytváření animací pracuje VR-man s uzly skupin dílů, které si vytvořil. Informace o animaci jsou uložena nad těmito uzly skupin (nad - z hlediska stromové struktury FHS). Program VR Aid musí změny FHS provádět pouze v podstromech uzlů skupin a zajistit, aby dříve vytvořená animace byla i nadále funkční. VRAid nesmí měnit části FHS a VRP stromu mimo uzly skupin.

Kopírování dílů v rámci FHS souboru

Některé díly mohl VR-man být v jednom FHS souboru použít vícekrát. FHS neumí pracovat s odkazy v rámci jednoho souboru (resp. tato funkcionalita VD2 je zatížena chybami a nepoužívá se). Při nahrazování dílů za nové verze musí VRAid nahradit všechny výskyty dílu (všechny výskyty v jedné verzi prezentace).

Zrcadlení dílu

Některé díly jsou vykoustruovány pouze pro jednu stranu automobilu, protože druhá strana je zrcadlovým obrazem. Při nahrazování dílů musí VRAid nahradit i tyto zrcadlené díly (mají koncovku jména dílu .mirror) a opět správně je zrcadlit.

Materiály

Při tvorbě prezentace přiřazuje VR-man dílům materiály, které určují, jak bude zobrazen povrch dílu. Při nahrazování dílů musí VRAid být tyto materiály použít i u odpovídajících nových dílů.

2.3 Návrh uživatelského rozhraní

Aplikace VR Aid by neměla provést celou úpravu sama bez zásahu uživatele. Problém práce VR-mana není definovaný tak přesně, aby bylo možné vytvořit plně automatickou aplikaci, která by přinesla očekávaný výsledek. Je možné, že by VR-man při opravách (plně automatického) neočekávaného výsledku strávil víc času, než když by žádnou pomůcku nepoužil.

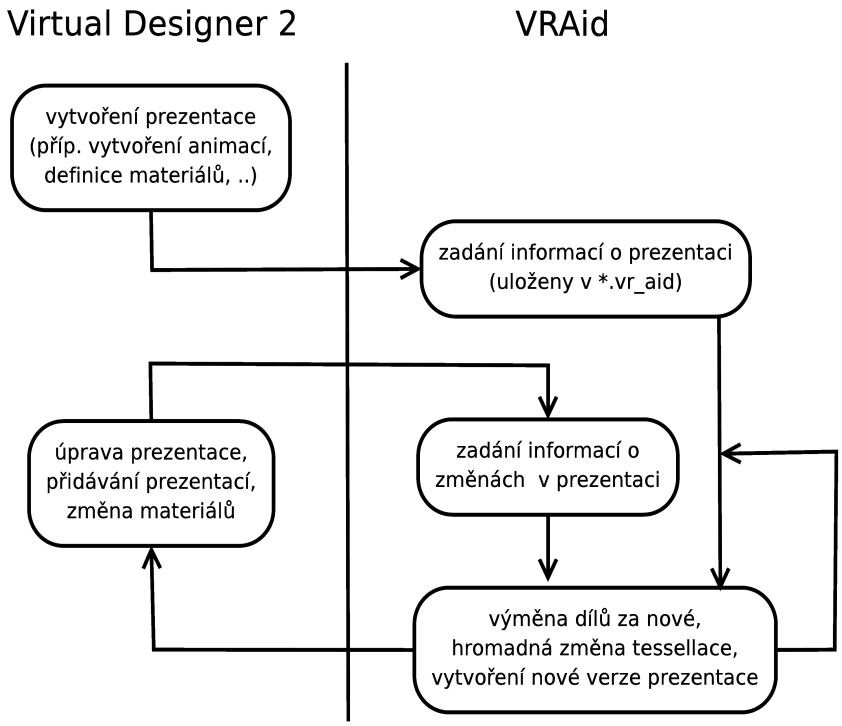
Uživatelské rozhraní by mělo být navrženo tak, aby VR-man v každé fázi věděl, co může s aplikací dělat a v jakém stavu je rozpracovaná verze prezentace. VR-man by měl být možnost naplánované změny uložit a provést je někdy jindy.

3 Rozpracování vybraného přístupu

3.1 Spolupráce VRAid a VD2

Z analýzy pracovních postupů VR-manů jsem viděl, že VD2 je v mnoha oblastech práce vynikajícím nástrojem a je nad možnosti diplomové práce kopírovat tuto funkcionalitu. Na druhou stranu VD2 při několika málo úkonech selhává a práce s ním je neohrabaná. Ideální by samozřejmě bylo tuto funkcionalitu do VD2 doprogramovat, bohužel se ale jedná o komerční aplikaci s uzavřeným zdrojovým kódem. Nenašel jsem žádnou možnost, jak potřebné funkce do VD2 přímo přidat.

Druhou nejlepší možností (pro kterou jsem se nakonec rozhodl) bylo vytvořit program, který bude s VD2 těsně spolupracovat. VR-man vytvoří prezentaci ve VD2, tam také definuje materiály, naskriptuje animace. Potom bude moci použít program VRAid pro záměnu starých dílů za nové, pro vytvoření nové verze nebo pro hromadnou změnu tessellace dílů. A potom bude moci VR-man zase pracovat s tou samou prezentací (tím samým souborem) se VD2 a upravovat animace, přidávat další, měnit materiály nebo přidávat další díly. Průběh (dlouhodobé) práce by mohl vypadat nějak takto:



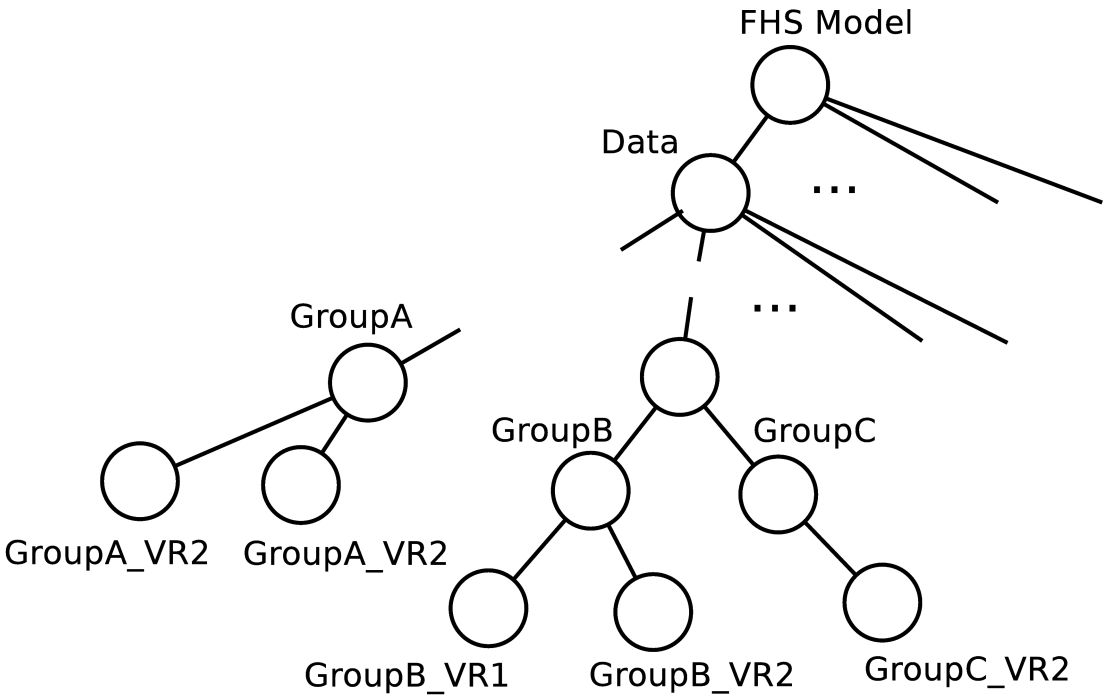
Obrázek 4: Průběh práce s aplikacemi VRAid a VD2

3.2 Dohoda s VR-many

Pro to, aby spolu mohly VD2 a VRAid spolupracovat, je třeba vymyslet konvence, které bude splňovat prezentace. Tyto konvence budou VR-mani dodržovat a následně je bude moci využít VRAid. Dohoda s VR-many vychází z převažující praxe na pracovišti virtuální techniky. VR-manů by nemělo činit potíže dohodu dodržovat.

3.2.1 FHS část prezentace

VR-mani budou udržovat prezentaci ve tvaru dle obrázku 4.



Obrázek 5: Příklad FHS stromu prezentace (2 verze - VR1 a VR2)

Definice: Uzel skupiny (dílů)

Při práci VR-man pracuje se skupinami dílů. Uzel skupiny je nejbližší předek společný celé skupině dílů (z hlediska stromu FHS). Název uzlu může obsahovat alfanumerické znaky a podtržítka.

Na obrázku 5 uzly skupin představují kolečka pojmenovaná GroupA, GroupB, GroupC. Uzel skupiny samozřejmě nemůže být předkem jiného uzlu skupiny.

Definice: Název verze

Verze prezentací nejsou číslovány. Verze mají jméno, ze kterého může, ale nemusí být zřejmé, v jakém jsou pořadí jsou verze za sebou. Jméno verze může obsahovat alfanumerické znaky (ne podtržítko).

Při práci s verzemi není moc důležité, jaká verze následovala po které. Je odpovědností VR-mana, aby si v této záležitosti udržel pořádek. Pro program VRAid je důležité jen, jak se jmenuje aktuální verze a případně jak se má jmenovat verze nově vytvářená. Na obrázku názvy verzí představují VR1 a VR2.

Definice: Uzel verze

Uzel verze je přímým potomkem uzlu skupiny. Jméno uzlu verze se skládá ze jména uzlu skupiny a názvu verze spojené podtržítkem.

Sourozencem uzlu verze mohou být pouze uzly jiných verzí, ale stejné skupiny dílů. Na obrázku 5 uzly verze představují kolečka pojmenovaná GroupA_VR1, GroupA_VR2, GroupB_VR1, GroupB_VR2, GroupC_VR2.

3.2.2 VRP část prezentace

VR-mani budou udržovat strukturu variant ve VRP podobně jako v FHS. Důležitou částí jsou variant sety. Jméno variant setu musí začínat jménem verze.

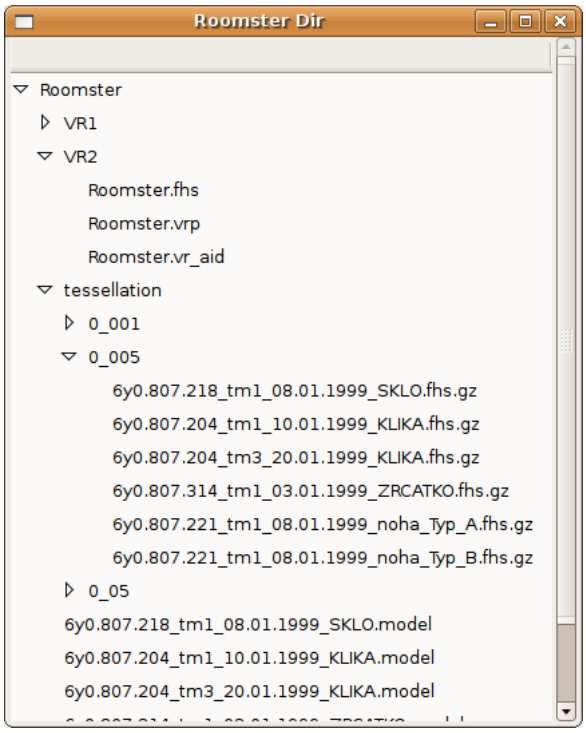
3.2.3 uložení souborů na disku

VR-mani budou ukládat soubory na disk následujícím způsobem. Pro každou prezentaci budou mít vytvořený adresář.

Definice: Adresář prezentace

Adresář, do kterého se ukládají všechny soubory související s konkrétní prezentací. Obsahuje ještě nepřeložené díly, přeložené díly roztříděné podle tessellace a zvlášť uloženou každou verzi prezentace (každá verze navíc umožňuje přepínat na verze minulé).

Adresář na disku vypadá takto:



Obrázek 6: Struktura adresáře prezentace

Definice: Adresář verze (prezentace)

Adresář, ve kterém je uložena prezentace s určitou nejnovější verzí. Název adresáře verze odpovídá názvu této nejovější verze.

Na obrázku 6 jsou vidět 2 adresáře verze - VR1 a VR2.

Definice: Adresář tessellace

Adresář, který obsahuje díly přeložené s určitou velikostí tessellace. Název adresáře je [velikost tessellace s podtržítkem místo desetinné čárky]. Všechny adresáře tessellace jsou v adresáři [adresář prezentace]/tessellation.

Na obrázku 6 jsou vidět 3 adresáře tessellací 0_001, 0_005, 0_05.

Dle dohody konvertované soubory v adresářích tessellací mají stejný název (až na příponu) jako jim odpovídající soubory v adresáři prezentace.

3.2.4 Pojmenování dílů

Definice: Jméno dílu

Název uzlu dílu v prezentaci (s koncovkou původního souboru) nebo jmého souboru dílu na disku (také včetně koncovky souboru).

Dle dohody s VR-many bude jméno dílu konzistentní v prezentaci, v souborech exportovaných z KVS i v souborech prelozených Operou. VR-man tedy může při exportu z KVS pojmenovat díl libovolně, pak ale musí toto jmého zachovávat a používat i v dalších fázích tvorby prezentace.

3.2.5 Současné používání VRAid a VD2

VR-man musí zajistit, že při provozu programu VRAid nebudou ostatními programy měněny FHS soubory v adresáři prezentace. Ostatní programy mohou soubory přidávat, ale nesmí stávající soubory měnit. Je to proto, že program VRAid spoléhá na to, že data FHS uzlů může kdykoli znovu načíst z disku. Po ukončení VRAid (resp. otevření jiné prezentace) je možné FHS soubory měnit libovolně.

3.3 Informace spravované programem VRAid

Nejdůležitější informací, která programu VRAid vůbec umožňuje pracovat, je seznam názvů verzí. Už jen z názvu verze je schopen získat příslušné uzly verzí - uzly s příponou rovnou názvu verze (s tím, že uzel verze nemůže být potomkem jiného uzlu verze).

VRAid uchovává informace o parametrech jednotlivých dílů. Tyto informace mohly být získány přímo ze jména dílu, ale také je mohl zadat/pozměnit VR-man (např. díl nemusel být pojmenovaný na základě parametrů v KVS). Pak je nutno tyto informace uchovat, aby je VR-man nemusel v budoucnu zadávat znovu.

VRAid uchovává informace o akcích, které VR-man chce s díly udělat. Např. je k dispozici více nových verzí dílu, ale VR-man nechce použít tu úplně nejnovější a zadá jinou. Nebo vůbec nechce použít novou verzi dílu.

3.4 Přechod od práce ve VD2 k práci ve VRAid

3.4.1 První zadávání informací o prezentaci

VR-man musí programu VRAid pomoci upravit prezentaci dle dohody s VR-many.

Pokud již prezentace splňuje dohodu s VR-many, stačí označit uzly skupin. VRAid pak může zjistit přípony uzlů verzí (přímí potomci) a vytvořit seznam názvů verzí.

Pokud ještě prezentace dohodu s VR-many nesplňuje, označí VR-man uzly skupin (ty už v prezentaci být musí) a zadá název první verze. VRAid pak vytvoří uzly verzí jako přímé potomky uzlů skupin (poduzly uzlů skupin budou teď poduzly uzlů první verze). Zároveň vytvoří soubor VRP a první variant set, který aktivuje první (a zatím jedinou) verzi.

3.4.2 Další zadávání informací o prezentaci

Program VRAid do značné míry spoléhá na to, že získá parametry dílu ze jména uzlu v FHS nebo jména souboru na disku. Může se však snadno stát, že parametry dílu se VRAid nepodaří získat, případně že získá parametry špatné. Je potřeba, aby VR-man mohl získané parametry dílů zkontrolovat, změnit, nebo zadat úplně nové.

VRAid musí získat parametry stávajících dílů. O to se pokusí automaticky a nechá VR-mana výsledek odsouhlasit.

Definice: Použitý/Stávající díl

Díl, který již v prezentaci existuje.

VRAid musí také získat parametry nových dílů na disku. O to se opět pokusí automaticky a nechá VR-mana výsledek odsouhlasit.

Definice: Nový díl

Díl, který je zatím jen na disku a v prezentaci ještě nebyl použit

Takovému automatickému získání parametrů dílu s odsouhlasením VR-mana říkám identifikace dílu.

Definice: Identifikace dílu

Spárování jména souboru (resp. jména FHS uzlu) a parametrů, které odpovídají tomuto dílu v KVS

Po identifikaci dílů už lze s VRAid pracovat; generovat nové verze prezentací nebo hromadně měnit tessellaci.

3.5 Přechod od práce ve VRAid k práci ve VD2

Při ukončení práce ve VRAid a přechodu k práci ve VD2 nejsou žádné zvláštní úkony potřeba.

4 Diskuse použitého řešení vzhledem k původnímu zadání

V průběhu práce na programu VRAid jsem stále více poznával prostředí VR-manů a jejich potřeby. Některé části původního zadání se ukázaly jako důležité a pro prácí VR-manů užitečné, jiné se ukázaly jako kontraproduktivní. Např. replikování funkcionality, která je již skvěle implementována v programu VD2 za kontraproduktivní považuji. Nová implementace stejné funkcionality by v nejlepším případě VR-many mátla.

Zároveň se také měnily podmínky ve Škodě Auto. Prostředky, které měla poskytnout Škoda Auto a se kterými se při zadávání diplomové práce počítalo, se po dlouhé době ukázaly jako nedostupné.

Musel jsem se rozhodnout mezi tím, zda vyhovím formálnímu zadání diplomové práce, a tím, zda poskytnu Škodě Auto užitečný program. Rozhodl jsem se přizpůsobit změněným podmínkám.

4.1 Původní zadání diplomové práce

Uvádím zadání diplomové práce v původním znění.

Vytvořte programový systém, který umožní prezentovat komplexní trojrozměrné objekty ve virtuálním prostředí. Při jeho vývoji spolupracujte s odborníky ze společnosti Škoda Auto.

Cílem práce je automatizovat převod CAD modelů do tvaru vhodného pro interaktivní prezentaci v reálném čase, a to v prostředí společnosti Škoda Auto. Analyzujte možnosti řešení, navrhněte vhodný postup a implementujte program, obsahující následující vlastnosti a funkce:

- rozhraní vzhledem k databázi CAD modelů,
- využívání systému konverze CAD dat,
- zpracování formátu FSH systému Virtual Design 2,
- vytváření knihoven materiálů,
- ukládání mezivýsledků v různých fázích procesu, jejich opakované využívání,
- uživatelské rozhraní dle zvyklostí firmy Škoda Auto,
- správa uživatelů a skupin,
- správa ručně rozkládaných dílů,
- schopnost opakovaně využít uživatelské změny ve výsledném trojrozměrném modelu (animace, zvuky)

Při řešení využijte vhodnou databázi. Program realizujte tak, aby byl nezávislý na platformách (IRIX, Linux, Windows).

V dalších kapitolách budu postupně diskutovat jednotlivé části zadání s tím, jakým způsobem byly implementovány, připadně jaké jiné řešení problému bylo použito a proč.

4.1.1 Vytváření prezentací z CAD modelů

Postup VR-mana při počátečním vytváření prezentace jsem popsal v kapitole 2.1.1 . Pro VR-mana se jako ideální jeví stávající postup; používání Programu VD2.

4.1.2 KVS - databáze CAD modelů

Požadavky na aplikace VR Aid se od začátku vývoje upřesňovaly, ale žádný z nich neměl na průběh takový vliv jako rozhraní KVS.

Databáze KVS je velice sofistikovaná a rozsáhlá. Jsou v ní uloženy úplně všechny CAD modely (a související dokumenty) celé Volkswagen Group. Pro KVS je k implementováno několik rozhraní:

- webové HyperKVS
- samostaný program HyperMona s GUI
- řádkový kvsclient32 pro použití z příkazového řádku
- RPC
- CORBA

Z těchto rozhraní mi připadaly použitelné kvsclient32, RPC a CORBA. Tyto 3 rozhraní jsou však ve ŠKODA AUTO, a. s. zakázány. V minulosti totiž docházelo k zahlcení databázových strojů kvůli nevhodně zadaným dotazům. Přístup k těmto rozhraním mají pouze vyhrazené aplikace, kterým musí být nejprve přidělen identifikační kód. Tyto aplikace stále potřebují pro přístup do KVS autentizaci uživatele, který je v dané chvíli používá, ale kromě toho musejí mít ještě tento identifikační kód pro aplikace.

Kód pro aplikace přiděluje administrace KVS v Německu a během práce na programu VRAid (od března 2006) se tento kód nepodařilo získat. Na jaře 2007 se administrátorovi v Mladoboleslavské pobočce podařilo získat formulář, podle kterého by získání přístupového kódu mělo výt rutinní záležitostí. Kromě tohoto světlého momentu byla ostatní komunikace bezvýsledná. Německá administrace KVS nikdy nevydala záporné stanovisko, takze příčiny selhání žádosti o přístupový kód jsou dodnes nejasné.

Pokoušel jsem se použít i jiný způsob komunikace s KVS. Bohužel moje pokusy o napsání webového robota pro HyperKVS selhaly. HyperKVS obsahovalo množství JavaScriptu, přesměrování a dalšího záhadného chování. Nepodařilo se mi napsat program, který by dokázal vyhledat a stáhnout CAD model z KVS.

V létě 2007, kdy už byla situace kolem diplomové práce kritická, jsem se musel rozhodnout myšlenku propojení s KVS opustit a program VRAid koncipovat jinak. VR-man bude muset soubory stahovat s KVS sám (lze hromadně - viz kapitola 2.1) a program VRAid bude pracovat s těmito staženými soubory na disku.

4.1.3 správa uživatelů a skupin

Podle původního návrhu měl program VRAid stahovat CAD data z databáze KVS. Součástí měla být také cache CAD souborů. Přístup k této cache měl být kontrolovaný podobně jako je kontrolovaný přístup zaměstnanců Škoda Auto do KVS. Ve Škodě Auto je všechno tajné, zejména nové návrhy automobilů a nic se nesmí dostat do nepovolaných rukou. Z tohoto důvodu jsem měl do VRAid zapracovat i správu uživatelských práv a přístupu ke CAD dílům.

Vzhledem k tomu, že jsem KVS nakonec nemohl použít, stala se cache nepotřebnou a s ní i správa přístupu ke cachovaným CAD dílům.

4.1.4 využívání systému konverze CAD dat

Pro koverzi dat z formátu Catia nebo Iges používá VR-man programy Opera nebo DeltaGen. Oba tyto programy je možné používat z příkazové řádky. Podle původního záměru

měl program VRAid přímo volat Operu a zajišťovat překlad dílu se správnou tessellací.

Později jsem zjistil, že VR-mani používají dávkové soubory (které volají Operu) pro hromadnou konverzi dílu do formátu FHS. Zapracování volání Opery do programu VRAid by VR-manům nějaký čas ušetřilo, ale ne mnoho.

Nakonec je tato funkcionalita zajištěna spoluprací VR-mana. Ten při hromadném konvertování dílů ukládá přeložené díly do adresářů podle toho, s jakou tessellací jsou díly konvertovány (viz dohoda s VR-many, kapitola 3.2.3).

4.1.5 zpracování formátu FSH systému Virtual Design 2

Ten to bod zadání jsem v programu VRAid plně implementoval. Program VRAid dokáže (v rámci svých potřeb) s formátem FHS pracovat mimořádně rychle. Načítání a počáteční analýzu souboru FHS zvládne VRAid ve třetinovém čase oproti komerčnímu VD2. Na jednu stranu program VD2 jistě formát FHS analyzuje podrobněji, na stranu druhou program VRAid je napsán v interpretovaném jazyku Python (jedna krátká funkce přepsána do C).

Kromě formátu FHS jsem implementoval i formát VRP. Program VRAid umí pracovat s formátem VRP, ale více pracuje jen s částmi, které řídí varianty a variant sety.

4.1.6 vytváření knihoven materiálů

Při zadávání diplomové práce jsme s Ing. Červeným pokládali za užitečné, že by si VR-man mohl při tvorbě prezentace vybrat z knihovny předpřipravených materiálů. Po bližším seznámení se s postupy VR.manu jsem ale musel tento názor změnit. VR-man mění materiály v prezentaci velmi často a přizpůsobuje parametry materiálů tak, aby v dané prezentaci vypadaly co nejlépe. K tomu používá program VD2, který umí materiály měnit hromadně (VD2 umí vybrat všechny uzly se stejným materiálem). Nebyl tedy žádný důvod knihovny materiálů vytvářet.

4.1.7 opakované využívání práce VR-mana

Tento požadavek řeší program VRAid tak, že dokáže pozměnit prezentaci aniž by zničil práci, kterou již VR-man udělal. Při nahrazování dílů za nové verze zachovává materiály, které VR-man dílům přiřadil, případně aplikuje stejné transformace.

VRAid dokáže vytvořit novou verzi prezentace a přitom upravit varianty tak, aby nové verzi odpovídaly a obsahovaly všechny změny, které VR-man provedl již ve verzi předchozí.

4.1.8 uživatelské rozhraní dle zvyklostí firmy Škoda Auto

Uživatelské rozhraní jsem vytvářel podle filozofie, že uživatel by měl mít před očima vždy pouze ty ovládací prvky, které potřebuje. Rozdělil jsem proto práci potřebnou pro zadávání dat a generování nových verzí prezentace do několika myšlenkově ucelených dialogů. Tyto dialogy jsou blíže popsány v kapitolách 6.2 - 6.15 .

4.1.9 správa ručně rozkládaných dílů

Některé díly jsou konstruovány jako jeden celek, ale přitom se skládají z několika částí vyrobených z různých materiálů. VR-man musí tyto díly po stažení z KVS rozdělit na několik částí, aby je mohl v prezentaci správně zobrazit.

Při počátečním návrhu aplikace jsme s Ing. Červeným zamýšleli, že záměna dílů za nové bude vysoce automatizovaná. Že VRAid sám zjistí nejnovější verzi dílu, stáhne ji z

KVS, přeloží Operou a začlení do prezentace. Přitom by bylo nutno umožnit uživateli zasáhnout do procesu a ručně (pomocí CAD programu) rozdělit patřičný díl.

V průběhu poznávání práce VR-manů jsem zjistil, že úlohu nelze takto automatizovat. VR-man musí do procesu zasahovat mnohem častěji. Navíc po vyřazení KVS z implementace je tento původně automatický úkon (stahování z KVS) ponechán na VR-manovi. VR-man tedy může v rámci stahování dílů rovnou díl i rozložit. Tato činnost nijak nenaruší užitečnost programu VRAid. Funkcionalitu VRAid jsem pro splnění tohoto požadavku vůbec měnit nemusel.

4.1.10 uživatelské změny v prezentaci - animace, zvuky

VR-man může do prezentace přidávat kromě geometrických dat, také další prvky virtuální reality. Např. animace vytvoří tak, že definuje transformaci v čase pro nějakou část FHS stromu. Pro zachování takové animece stačí, aby byla definovaná mimo uzly skupin a neodkazovala dovnitř uzlů skupin (přímo na uzel skupiny ukazovat může). Při animaci menších částí automobilu musí VR-man vytvořit uzel skupiny pro každou takovou část zvlášť. Tento postup odpovídá dosavadní praxi VR-manů při vytváření animací a je součástí dohody s VR-many (kapitola 3.2.1).

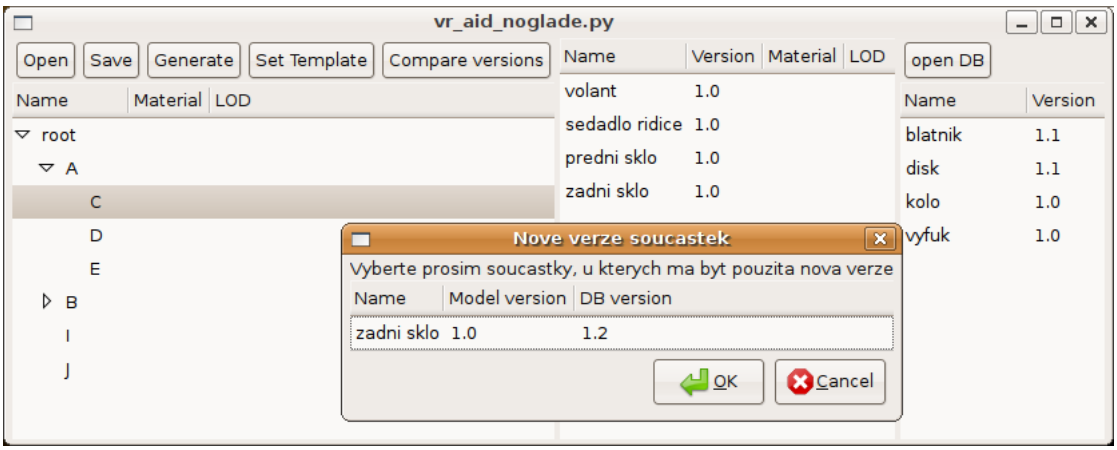
Veškeré transformace FHS stromu v důsledku změny verzí dílů jsou tak odděleny od tvůrčí práce VR-mana. Při generování nové verze nebo při hromadné změně tessellace jsou uživatelské změny zachovány.

4.2 Fáze vývoje

V této části popisuji i slepé vývojové větve. V prvních dvou verzích jsem rozhraní do KVS simuloval, abych vůbec mohl program vyvíjet.

4.2.1 verze 1

1. verze aplikace vznikla po intenzivní spolupráci s Ing. Červeným na počátku roku 2006. Soustředil jsem se tehdy na problém jednoduchého vytváření prezentací a zcela automatickou záměnu starých verzí dílů za tu nejnovější nalezenou v KVS. Tehdy jsem počítal s rozhraním do KVS a s tím spojeným vytvořením serverové části, databází uživatelů, jejich autentizace pomocí KVS, správou skupin uživatelů a kešováním dílů stažených z KVS.



Obrázek 7: Ukázka dialogu pro vyhledávání a záměnu nových dílů. VR Aid verze 1

4.2.2 verze 2

Po několika konzultacích s VR-many jsem nakonec celou aplikaci přepsal. Vyšlo najevo, že při vytváření zcela nové prezentace používají VD2 a že při tom nedochází k žádným problémům a časovým ztrátám. Dále se ukázalo, že ještě před skutečnou záměnou dílů dostane VR-man kusovník ve formátu XLS. V kusovníku jsou popsány všechny díly a jejich verze.

Tato verze aplikace tedy měla pouze automaticky zaměňovat staré díly v FHS za nové. K tomu měla využívat jako vstupu právě kusovníku, který obsahoval informace o nových dílech. Samozřejmě měla také stahovat tyto nové díly z KVS, překládat je programem Opera.

Bohužel se v pozdější fázi vývoje ukázalo, že kusovník VR-man nemá k dispozici vždy a aplikace jej tedy pro svoji funkci nemůže vyžadovat.

4.2.3 verze 3

Ke 3. fázi došlo v době, kdy již nebylo možno doufat v získání přístupu do KVS. Bylo třeba vymyslet způsob, jakým zefektivnit práci VR-manů a zároveň se bez přístupu do KVS obejít. Specifikace a důkladný popis 3. verze programu je obsahem celé této práce.

5 Prostředky používané při vývoji

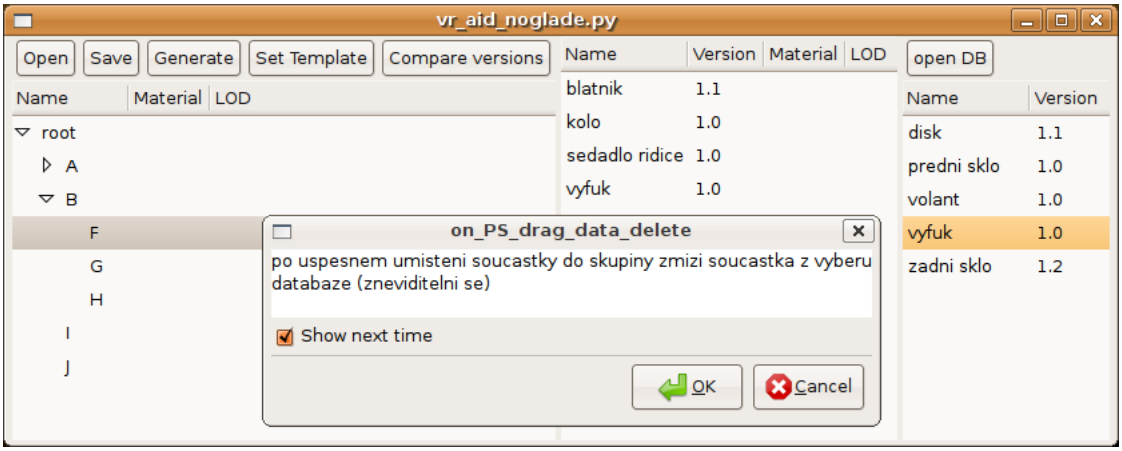
Hned při prvních kontaktech ve ŠKODA AUTO, a. s. se ukázalo, že tato práce bude připomínat skutečnou zakázku. Při shromažďování specifických požadavků na aplikaci VR Aid mi pomáhali zejména Ing. Leoš Červený, Ph. D. Antoním Míšek a ostatní VR-mani.

Protože ve společnosti ŠKODA AUTO, a. s. podléhá vše přísnému utajení, bylo možné testovat aplikaci VR Aid (kromě zcela jednoduchých příkladů) pouze osobně v Mladé Boleslavi. Jako (částečné) řešení tohoto problému byl při komunikaci s Ing. Červeným vytvořen systém *Messages*.

5.1 Messages

Při komunikaci s Ing. Červeným se objevil problém, jakým způsobem sdělit druhé straně požadované (nebo zamýšlené) vlastnosti aplikace. Obyčejný text se ukázal jako neefektivní a časté dojíždění do Mladé Boleslavi bylo problematické. Vznikla potřeba vytvořit nástroj, který by pomohl specifikovat akci provedenou v uživatelském rozhraní a popsat odpovídající reakci na tuto akci. Navíc by měl umožnit dát tyto 2 popisy co nejintuitivnějším způsobem do souvislosti.

Odpovědí na tento problém byl systém *Messages.* Použití je následovné: Kdykoli zákazník provede v uživatelském rozhraní nějakou akci, objeví se dialog s identifikátorem této akce a uživatelsky srozumitelným popisem, jak hodlá aplikace reagovat. Zákazník může tyto malé kousky dokumentace upravovat a ukládat na disk (jsou ukládány do jediného souboru). Programátor aplikace dokumentuje pomocí Messages části programového kódu. Na druhé straně zákazník, který obdrží takovouto dokumentaci, vidí hned při ovládání programu, co svými akcemi způsobí. Pokud si uživatel myslí, že by se aplikace měla chovat jinak, jednoduše změní text v dialogu a následně odešle soubor *Messages* zpět programátorovi. Programátor porovná došlý soubor *Messages* s dříve odeslaným a ihned zjistí reakci zákazníka, rozdílnost jeho názorů a oba spolu mohou domluvit další podrobnosti.



Obrázek 8: Ukázka použití systému Messages

Na světě existuje dnes spousta systémů pro sledování požadavků zákazníka: Bugzilla, Mantis, apod. Všechny mají ale jednu zásadní nevýhodu: zákazník se musí naučit způsob práce programátora, aby s ním byl schopen komunikovat. Navíc pro zákazníka je často obtížné popsat přesně celý problém.

Systém Messages toto řeší. Zákazník pouze jakoby používá program, který je pro něj vyvíjen. Přitom se dovídá, co program v kterém okamžiku dělá a může zaznamenat svoji

zkušenst a svoje přání. **Zákazník se nemusí starat o to, aby popsal, v jaké fázi používání programu zaznamenal problém**. Prostě jen problém popíše v okamžiku, kdy ho zaznamená. Programátor z textového souboru zjistí, co a v kterých fázích programu zákazník zapsal. V pozdější fázi vývoje může tento systém sloužit ke sledování chyb. I jednoduché sdělení zákazníka "nefunguje to" je programátorovi užitečné. Má totiž zákazníkovo sdělení automaticky spojené s akcí programu, kterou zákazník prováděl a může mnohem snadněji identifikovat problém.

5.2 Programovací jazyk

Jako programovací jazyk jsem zvolil Python protože s ním mám několikaleté kladné zkušenosti.

Python je jazyk, který poskytuje programátorovi vysoce abstraktní možnosti vyjadřování. Myšlenku algoritmu tak mohu uceleně vyjádřit na malém prostoru a je snadno pochopitelná. Při čtení programů v jazyce C jsem pro pochopení algoritmu často musel studovat několik stránek kódu (to zpomaluje práci).

Python je jazyk interpretovaný a tedy zdaleka ne tak rychlý jako kompilované jazyky (C) nebo částečně interpretované jazyky (např. Java). Na druhou stranu k Pythonu existuje spousta specializovaných knihoven napsaných v jazyce C, které odvedou výpočetně náročnou práci.

Díky jazyku Python jsem mohl vytvořit několik prototypů aplikace VRAid. Až při vytvoření prototypu totiž obvykle zákazník (v tomto případě Ing. Červený a VR-mani) pozná, jaké funkce potřebuje a jaké jsou naprosto zbytečné. Jazyk python umožňuje prototyp aplikace vyvinout rychle a ušetřil mi tím spoustu práce na (později zavržených) prototypech.

5.3 GTK, Glade

Konkrétně v programu VRAid velmi využívám knihovnu PyGTK pro programování grafického uživatelského rozhraní. Program VRAid je tak při používání velmi rychlý. Navíc knihovna PyGTK funguje na operačních systémech Windows i Linux. Knihovu lze získat v předkompilovaných balíčcích a tak jsem neměl problém s kompilováním pro každý operační systém zvlášť.

5.4 Demjson.py

Pro ukládání správy prezentací je použita knihovna demjson.py, jejímž autorem je Deron Meranda <http://deron.meranda.us/>. Knihovna slouží pro ukládání a načítání struktur ve formátu JavaScript Object Notation (JSON, rfc4627).

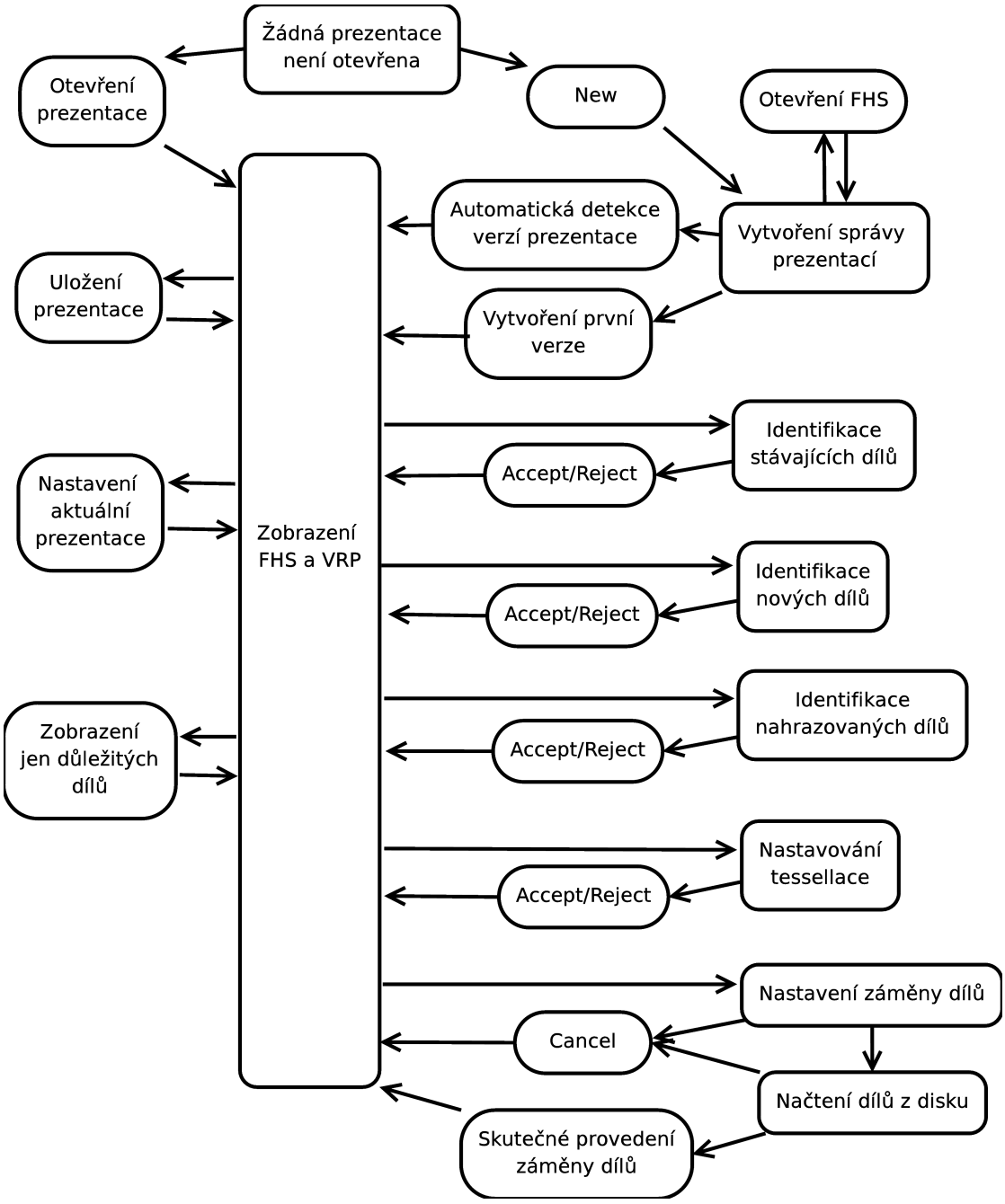
Ukázka JSON:

```
{
      "Image": {
          "Width":  800,
          "Height": 600,
          "Title":  "View from 15th Floor",
          "Thumbnail": {
              "Url":    "http://www.example.com/image/481989943",
              "Height": 125,
              "Width":  "100"
          },
          "IDs": [116, 943, 234, 38793]
```

6 Popis implementace

6.1 Stavový diagram práce s programem

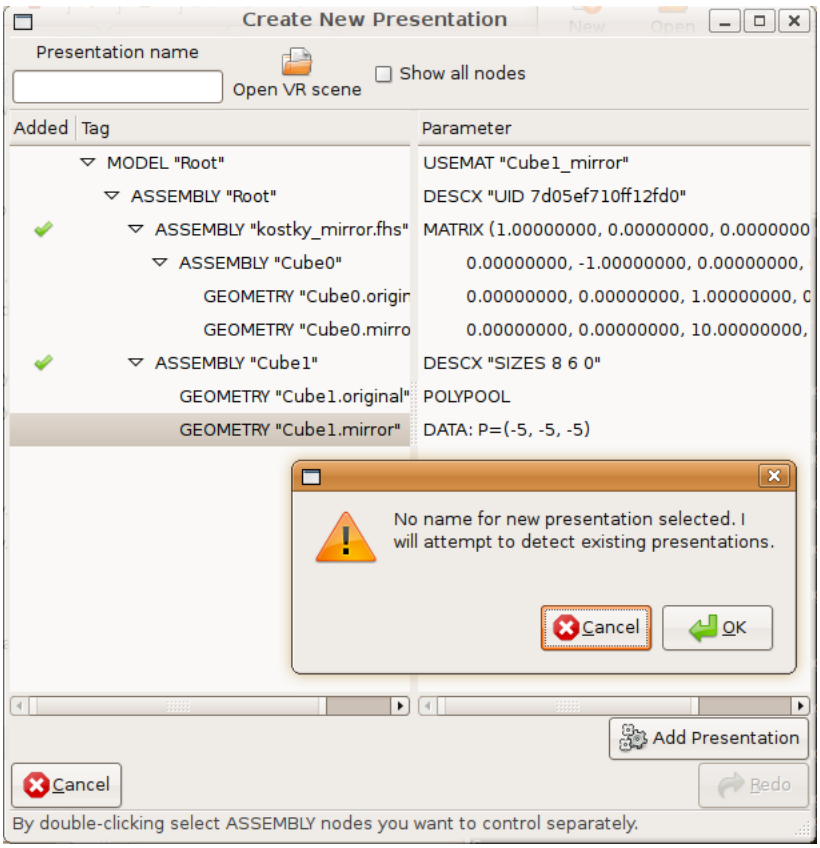
Program pomáhá uživateli vytvářet plán pro generování nové verze prezentace. Centrálním stavem aplikace je zobrazení aktuálního stavu prezentace (zobrazení variant setů, stromu variant, FHS stromu a parametrů jednotlivých FHS uzlů). Práce probíhá tak, že uživatel spustí nějakou akci, upraví část plánu a do tohoto centrálního stavu se vrátí. Pořadí akcí při tvorbě plánu závisí jen na uživateli, program mu nediktuje, kdy má co udělat.



Obrázek 9: Stavový diagram práce s VR Aid

6.2 První přechod od VD2 k VRAid - vytvoření nové správy prezentace

Uživatel může vytvořit novou správu prezentace (vr_aid) na základě prezentace v FHS souboru. Přitom FHS soubor může, ale nemusí splňovat dohodu s VR-many. Pokud dohodu s VR-many splňuje, je použit i přidružený soubor variant a VRAid ho znovu nevytváří.



Obrázek 10: Vytvoření správy prezentace

Uživatel najde a otevře soubor prezentace v FHS formátu.

Uživatel zadá ty FHS uzly, které představují uzly skupin (procházením FHS stromu a dvojklikem a příslušné uzly skupin).

Pokud uživatel zadá jméno verze prezentace, jsou mezi uzly skupin a jejich přímé potomky přidány nové FHS uzly (uzly verzí). Uživatel dále vybere uzel, který bude odpovídat kořenovému uzlu v souboru VRP. VRAid vytvoří odpovídající soubor VRP s jedinou variantou.

Pokud uživatel nezadá jméno prezentace, je mu nabídnuto rozpoznání existujících verzí prezentací a po odsouhlasení VRAid automaticky rozpozná existující verze a použije existující soubor variant VRP.

6.2.1 Použité algoritmy

Při vytváření první nové verze prezentace jsou uzly pojmenovány tak, že se jméno uzlu skupiny a jméno varianty spojí podtržítkem.

Automatické rozpoznání existujících verzí probíhá tak, že VRAid shromáždí zadané uzly skupin, vezme všechny jména potomků a vytvoří koncovky jejich jmen (při rozdělování

podtržítky). Jedinečné koncovky jmen tvoří seznam jmen verzí. Pro každou verzi vytvoří VRAid strukturu verze.

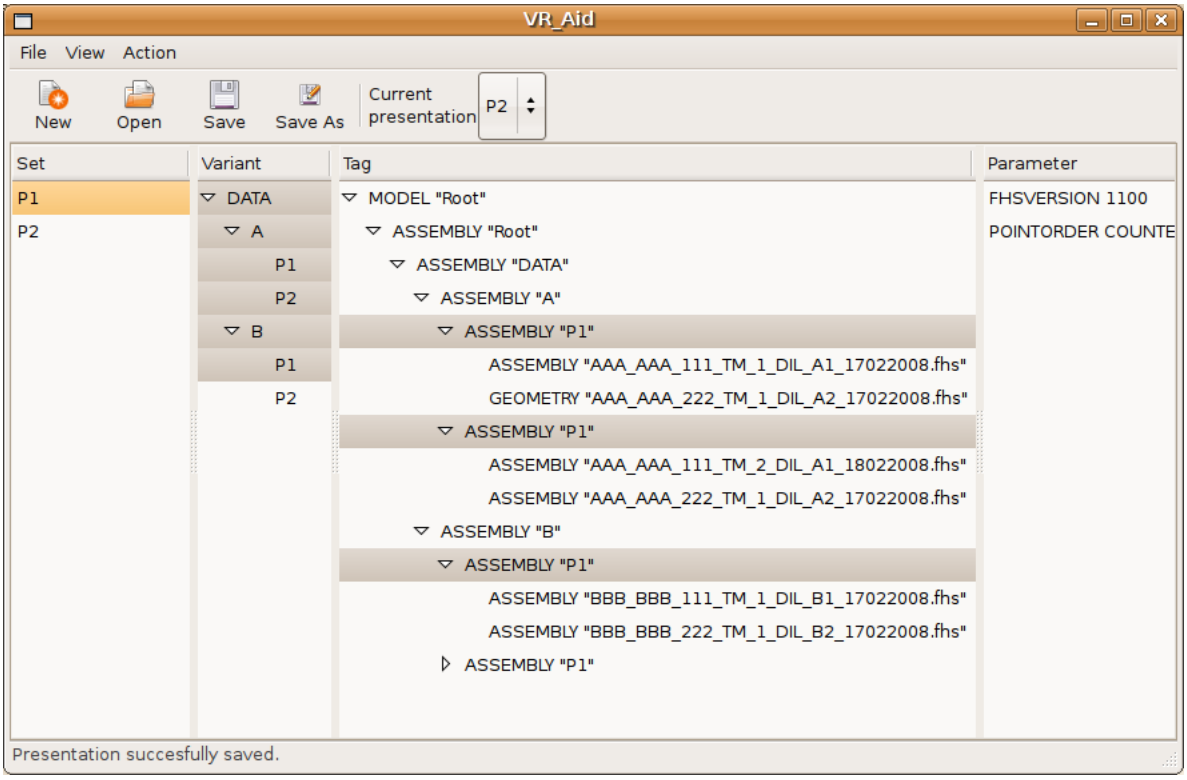
6.3 Uložení struktury správy prezentace

Uživatel může uložit strukturu správy prezentace (*.vr_aid). Spolu se strukturou správy prezentace se uloží pod stejným názvem souboru (s odpovídající koncovkou) FHS a VRP.

6.4 Otevření struktury správy prezentace

Uživatel může otevřít strukturu správy prezentace (*.vr_aid). Spolu se strukturou správy prezentace se otevře také dříve uložené FHS a VRP.

6.5 Zobrazení FHS a VRP



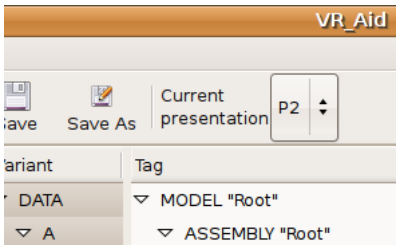
Obrázek 11: Zobrazení FHS, variant a variant setů

Uživatel vidí variant sety, varianty, FHS strom a parametry jednotlivých FHS uzlů.

Když uživatel klikne na variant set, VRAid označí všechny varianty, které zapíná tento variant set a zruší označení všech ostatních. Zároveň VRAid označí všechny FHS uzly, které odpovídají označeným variantám a zruší označení ostatních FHS uzlů.

Když uživatel klikne na variantu, VRAid označí všechny FHS uzly, které ovládá příslušná varianta a zruší označení ostatních FHS uzlů.

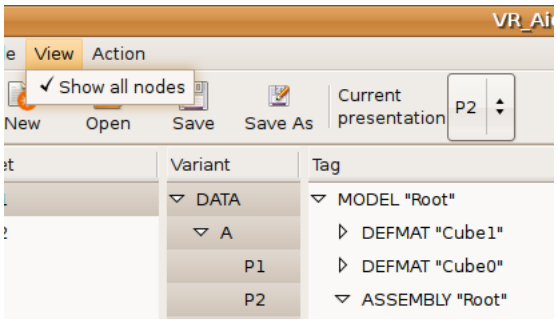
6.6 Nastavení aktuální verze prezentace



Obrázek 12: Nastavení aktuální verze prezentace

Uživatel si může vybrat aktuální verzi prezentaci. Tato volba určuje, se kterou verzí prezentace bude uživatel pracovat.

6.7 Zobrazení jen důležitých uzlů



Obrázek 13: Zobrazení jen důležitých uzlů

V základním režimu VRAid zobrazuje jen důležité FHS uzly. Jsou to hlavně FHS uzly ASSEMBLY a GEOMETRY, které patří aktuální verzi prezentace (samozřejmě jaké všichni jejich předci). Tato funkcionalita zpřehledňuje práci s VRAid.

Uživatel může tento základní režim vypnout a zobrazit všechny FHS uzly.

6.7.1 Použité algoritmy

Zobrazování uzlu

VRAid se při zobrazování FHS stromu rozhoduje u každého uzlu, zda ho zobrazí či ne.

Pokud tag FHS uzlu není v seznamu důležitých tagů, FHS uzel se nezobrazí.

VRAid zjišťuje, zda mezi předky FHS uzlu není uzel verze. Pokud není, uzel se zobrazí.

Pokud nalezený uzel verze odpovídá aktuální verzi prezentace, FHS uzel se zobrazí.

Jinak se uzel nezobrazí.

Detekce uzlu verze

Pokud jméno uzlu obsahuje jako koncovku některé ze jmen verzí a zároveň žádný jeho předek tuto podmínku nesplňuje, VRAid usoudí, že se jedná o uzel verze.

6.8 Automatická identifikace stávajících dílů

VRAid najde v podstromech uzlů aktuální verze všechny uzly dílů.

VRAid identifikuje nalezené díly. Pokud díl byl identifikován, označí ho jako použitelný (jinak jako nepoužitelný).

Použitelné i nepoužitelné díly přidá VRAid ke stávajícím dílům.

6.8.1 Použité algoritmy

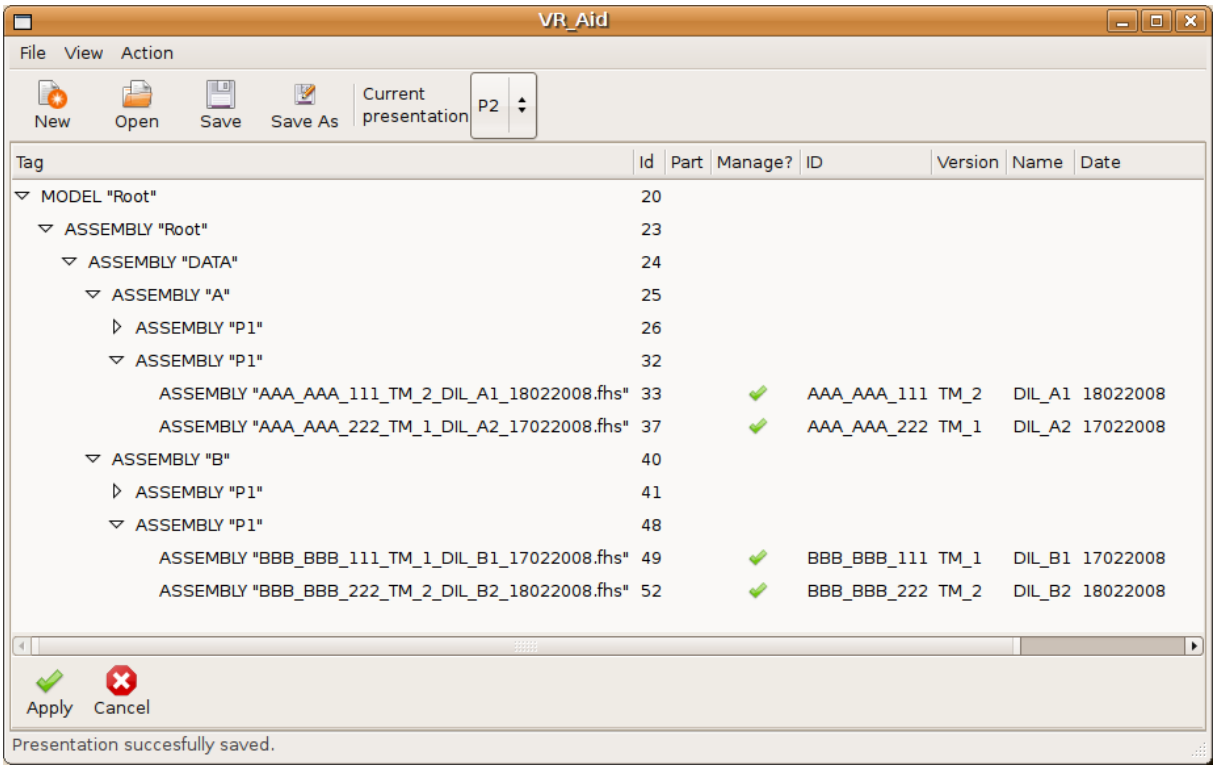
Nalezení dílů

VRAid detekuje uzly aktuální verze, projde jejich podstromy a všechny uzly se jménem končícím jako soubory FHS nebo CSB (s případnou další koncovkou .mirror) pokládá za díly.

Identifikace dílů

VRAid aplikuje na jména uzlů dílů regulární výraz. Pokud uspěje uloží získané parametry a díl je identifikován. Pokud neuspěje, díl není identifikován.

6.9 Identifikace stávajících dílů uživatelem



Obrázek 14: Identifikace stávajících dílů

Uživatel vidí strom FHS, označené stávající díly a jejich parametry. Může změnit parametry dílu (ručně identifikovat díl).

Uživatel může nastavit, zda stávající díl bude možné vyměnit (sloupec manage?).

Parametry dílů VRAid uloží do struktury verze.

6.10 Automatická identifikace nových dílů

VRAid najde v adresáři prezentace nové díly.

VRAid automaticky identifikuje nové díly.

Identifikované díly označí VRAid jako použitelné, jinak jako nepoužitelné.

Použitelné i nepoužitelné díly přidá VRAid do seznamu nových dílů.

6.10.1 Použité algoritmy

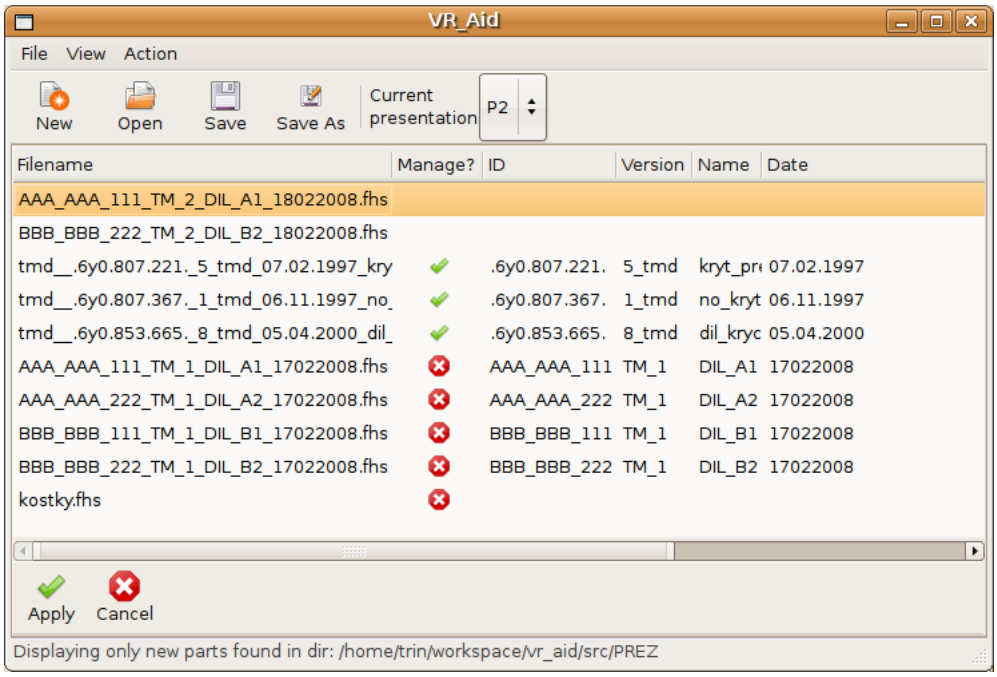
Nalezení nových dílů v adresáři prezentace

VRAid hledá v adresáři prezentace soubory s koncovkou FHS, CSB nebo nějakého CAD formátu. Nalezené soubory porovná se jmény stávajících dílů ve všech strukturách verzí. Za nové díly považuje ty, jejichž jména se v seznamech stávajících dílů nevyskytují.

Identifikace dílů

VRAid aplikuje na jména souborů regulární výraz. Pokud uspěje uloží získané parametry a díl je identifikován. Pokud neuspěje, díl není identifikován.

6.11 Identifikace nových dílů uživatelem



Obrázek 15: Identifikace nových dílů uživatelem

Uživatel vidí seznam dílů, jména jejich souborů, identifikované parametry a zda je díl použitelný či ne (sloupec manage?).

Díly jsou řazeny podle použitelnosti v pořadí "ještě nerozhodnuto", "použitelný", "nepoužitelný" a dále abecedně podle jména souboru.

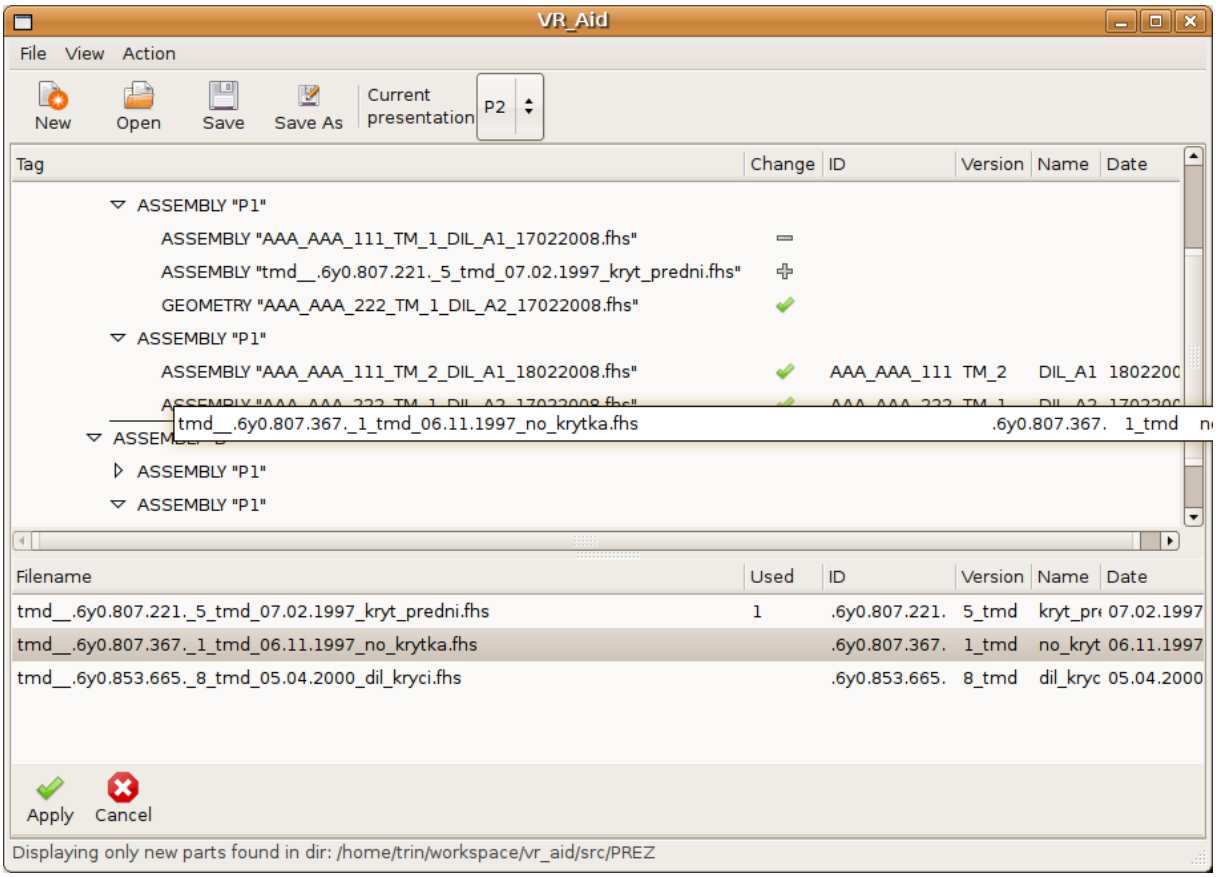
Uživatel může editovat parametry (ručně identifikovat díl) a měnit použitelnost dílu.

6.12 Automatické hledání nových verzí dílů

Aplikace prochází stávající použitelné díly a mezi novými použitelnými díly hledá nové verze. Pokud pro stávající díl najde nějaké se stejným ID a novější verzí nebo datem, vybere z těchto novějších díl s nejnovější verzí a datem (v tomto pořadí). Příslušnou dvojici starého a nového dílu zaznamená do struktury aktuální prezentace.

6.13 Identifikace nahrazovaných dílů

V tomto dialogu vytváří uživatel plán, které díly budou zaměněny za které. Jedná se pouze o vytvoření plánu, který ještě není proveden. Jde o to, aby uživatel viděl výsledky automatického hledání nových dílů a mohl odsouhlasit/změnit plán nahrazování.



Obrázek 16: Identifikace nahrazovaných dílů

Uživatel vidí FHS strom, použitelné stávající díly, jejich parametry a označení, zda bude tento díl vyměněn či nikoli. Uživatel dále vidí seznam nových použitelných dílů, jejich parametrů a informaci o tom, kolikrát jsou použity při záměně (sloupec used).

Uživatel si může nastavit (jako ostatně všude jinde) šířku sloupců. Šířky sloupců FHS stromu odpovídají šířkám sloupců v seznamu nových dílů.

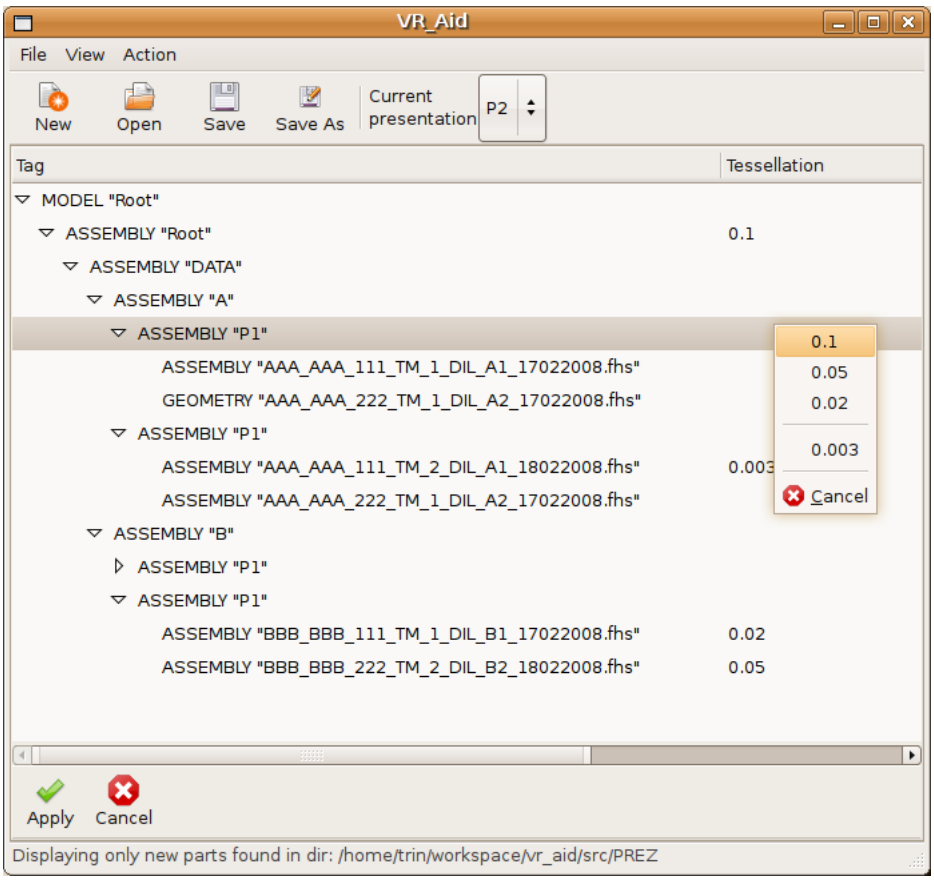
Díl může být označen jako odebíraný (mínus), přidávaný (plus) nebo beze změny. Odebíraný a přidávaný díle se vyskytují pouze spolu a označují výměnu dílu.

Uživatel může přetáhnutím nového dílu a jeho upuštěním na stávající díl určit, jaké díly budou zaměněny.

Uživatel může zrušit plánovanou záměnu dílů kliknutím pravého tlačítka myši a výběrem z menu.

6.14 Nastavování tessellace

Vyvolání pomocí Actions -> Tessellation



Obrázek 17: Nastavování tessellace

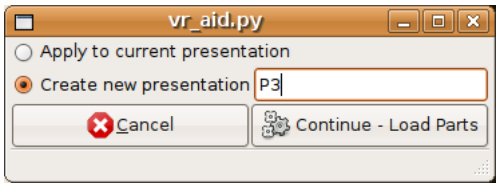
Uživatel vidí FHS strom a aktuální plánované tessellace.

Uživatel může plánovat tessellace kliknutím do sloupce Tessellation a zadáním tessellace.

Uživatel může plánovat a rušit tessellace kliknutím pravého tlačítka a výběrem z menu. V si může uživatel vybrat ze seznamu přednastavených tessellací a doplňujícího seznamu tessellací použitých v aktuální prezentaci.

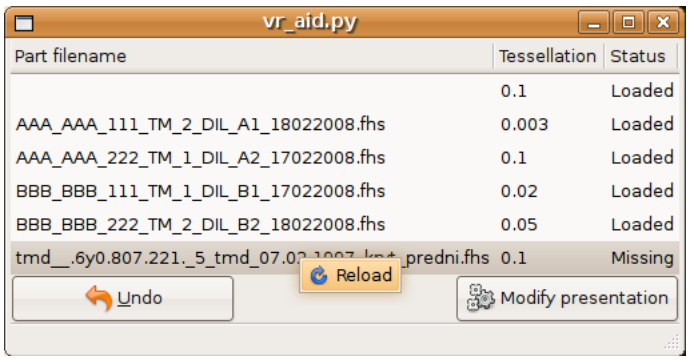
6.15 Záměna dílů

Vyvolání pomocí Actions -> Generate



Obrázek 18: Záměna dílů 1

Uživatel si může zvolit, zda nahrazování dílů proběhne v rámci aktuální prezentace nebo zda bude vytvořena prezentace nová. V tom případě musí novou prezentaci pojmenovat.



Obrázek 19: Záměna dílů 2

Uživatel vidí seznam dílů, které budou při záměně potřeba, jejich tessellaci a průběh načítání z disku.

Uživatel může zopakovat načítání dílu, pokud načítání selhalo.

Po pokynu "modify presentation" aplikace provede skutečnou záměnu.

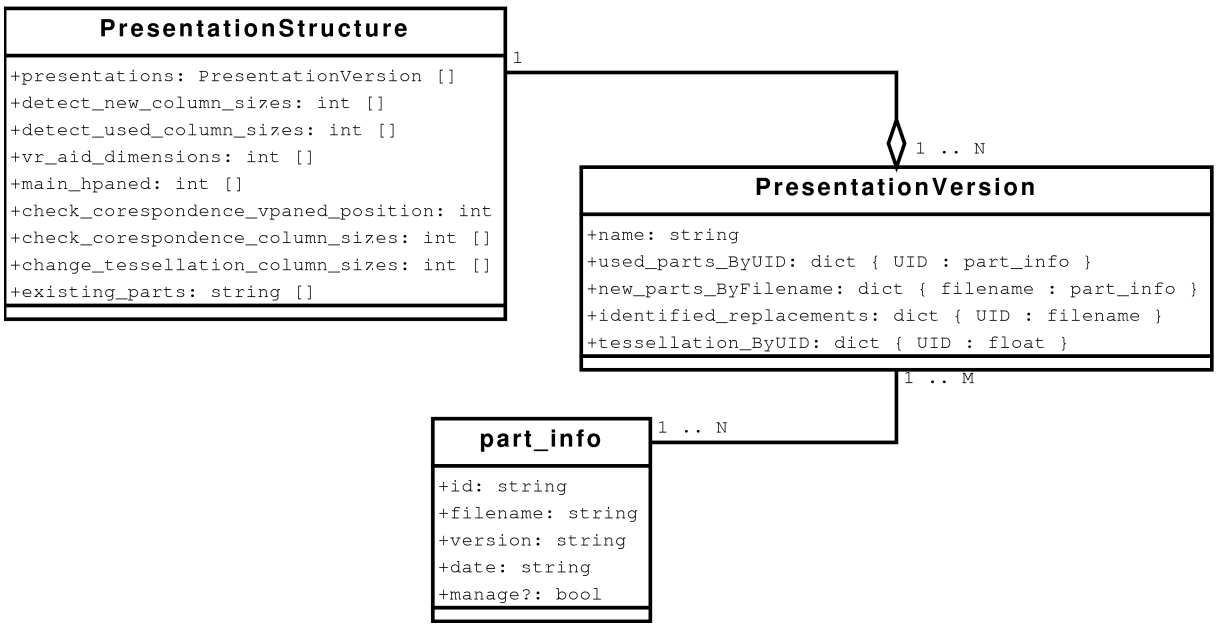
6.16 Struktura správy prezentace a popis formátu vr_aid

Struktura správy prezentace je centrální strukturou aplikace VR Aid a uchovávají se do ní všechny informace získané od uživatele. Ve struktuře jsou použita pouze čísla, textové řetězce, seznamy a asociativní pole. Neobsahuje žádná složitější propojení nebo odkazy a proto je možné přímo uložit celou strukturu na disk ve formátu JSON (více o formátu JSON v sekci 5.4). **Popis struktury správy prezentace je tedy zároveň popisem formátu vr_aid.**

Definice: Struktura (správy) prezentace

Struktura dat v paměti a na disku, kterou si VRAid o prezentaci udržuje

Struktura prezentace typicky spravuje několik verzí prezentace. Pro každou verzi je vytvořena identická struktura, která uchovává data získaná od uživatele (parametry dílů jsou uchovávány odděleně pro každou verzi). Struktura prezentace se navíc stará o uchování nastavení GUI - nastavení šířky jednotlivých sloupců, velikost okna aplikace.



Obrázek 20: datové struktury programu VRAid

6.16.1 PresentationStructure

Struktura prezentace. Obsahuje několik struktur (pro jednotlivé verze), název aktuální verze a spoustu nastavení GUI. Nastavení GUI se ukládá k prezentaci. To může být výhodné v případě, že VR-man pracuje na více prezentacích a tyto mají různé nároky na např. šířku sloupců. Struktura prezentace obsahuje následující položky:

presentations

seznam struktur verzí (PresentationVersion), popis Struktury verze níže

detect_new_column_sizes

šířky sloupců GUI při detekování nových dílů na disku

detect_used_column_sizes

šířky sloupců GUI při detekování stávajících dílů v prezentaci

vr_aid_dimensions

velikost okna aplikace VRAid

main_hpaned

šířky sloupců GUI v základním zobrazení prezentace (FHS strom, varianty, variant sety, informace o FHS uzlu)

check_corespondence_vpaned_position

výška horní části okna GUI při identifikaci nahrazovaných dílů

check_corespondence_column_sizes

šířky sloupců GUI při identifikaci nahrazovaných dílů

change_tessellation_column_sizes

šířky sloupců GUI při zadávání tessellace

existing_parts

Seznam jmen souborů stávajících dílů ve všech verzích prezentace. Používá se při identifikaci nových dílů. Zajišťuje, aby soubory stávajících dílů nebyly identifikovány jako nové díly.

6.16.2 Struktura verze

Struktura pro shromažďování informací specifických pro jednu verzi prezentace. Uchovává struktury part_info identifikovaných dílů (stávajících i nových), plánovaná nahrazení dílů a nastavení tessellace.

name

string - jméno prezentace

used_parts_ByUID

asociativní pole - klíčem jsou UID řetězce FHS uzlů, hodnotami jsou odpovídající struktury part_info

new_parts_ByFilename

asociativní pole - klíčem jsou jména dílů (dle souborů na disku), hodnotami jsou odpovídající struktury part_info

identified_replacements

plánované záměny dílů

asociativní pole - klíčem jsou UID stávajících dílů v FHS, hodnotami jsou jména nově identifikovaných dílů

tessellation_ByUID

plánované změny tessellace

asociativní pole - klíčem jsou UID uzlů v FHS (ne nutně dílů), hodnotami jsou desetinná čísla určující tessellaci.

6.16.3 Struktura dílu

Struktura part_info pro uložení parametrů o dílu.

id

string - ID dílu v KVS

filename

string - jméno dílu

name

string - popisek dílu, obvykle slovně vyjádřený název dílu

version

string - verze dílu ve formátu [celé číslo]_[string - typ verze]

Typ verze je při porovnávání verzí ignorován, může ho tvořit jakákoli posloupnost písmen.

date

string - datum vytvoření dílu ve formátu [den na 2 místa].[měsíc na 2 místa].[rok na 4 místa]

manage?

bool - True nebo False, povolení použití dílu při identifikaci nahrazovaných dílů a při samotném nahrazování

Příklad `part_info` převedeného do formátu JSON:

```
{
    "id" : "XXX.XXX.XXX",
    "filename": "tmd__.6y0.807.221._5_tmd_07.02.1997_kryt_predni",
    "name" : "kryt_predni",
    "version" : "5_tmd",
    "date" : "07.02.1997",
    "manage?":False
}
```

7 Popis formátů FHS a VRP

7.1 FHS

FHS slouží pro popis virtuálních scén v textovém formátu. Stromová struktura FHS vzdáleně připomíná VRML a funkcionalita se v mnoha místech překrývá. Kromě tohoto textového formátu existují i binární varianty FHB a FHS.GZ . Implementace těchto formátů není součástí této práce.

7.1.1 Popis formátu FHS

K formátu FHS poskytla společnost Škoda Auto, a. s. originální specifikaci. Velmi detailně popisuje všechna možná využití FHS, spoustu tagů. Popis jednotlivých tagů byl při vývoji knihovny FHS velmi důležitý, protože chování FHS často určují samotné tagy a nikoli obecné zákonitosti[1].

Ukázka FHS:

```
GEOMETRY "Cube1.mirror"
USEMAT "Cube1_mirror"
DESCX "UID 7d05ef710ff12fd0"
MATRIX (1.00000000, 0.00000000, 0.00000000, 0.00000000,
        0.00000000, -1.00000000, 0.00000000, 0.00000000,
        0.00000000, 0.00000000, 1.00000000, 0.00000000,
        0.00000000, 0.00000000, 10.00000000, 1.00000000)
DESCX "SIZES 8 6 0"
POLYPOOL
{
    P=(-5, -5, -5)
    P=(-5, -5, 0)
    P=(-5, 0, -5)
    P=(-5, 0, 0)
}
{
    POLYGON
    {
        I=1 I=3 I=2 I=0
    }
```

V ukázce jsou vidět nějaké *tagy* (`GEOMETRY`, `USEMAT`, `POLYGON`), jejich *parametry* (které mohou být víceřádkové a složené závorky, které ohraničují vnořené *uzly* nebo *data*. Některé tagy označují začátky uzlů (`GEOMETRY`, `ASSEMBLY`), některé začátek dat (`POLYPOOL, POLYGON)`. Dále je v ukázce zajímavý *tag* `DESCX` s *parametrem* UID. Hodnota UID jednoznačně identifikuje *uzel* v rámci souboru FHS a je použita při odkazování na *uzel* (viz formát VRP).

Knihovna FHS je psána tak, aby co nejvíce využívala obecnou strukturu FHS a k pozměňování svého chování na základě *tagů* jednotlivých uzlů se uchyluje jen v nutných případech. Například pro detekci *dat* se nepoužívají jména tagů, ale vyhodnocuje se struktura textu na začátku bloku {}.

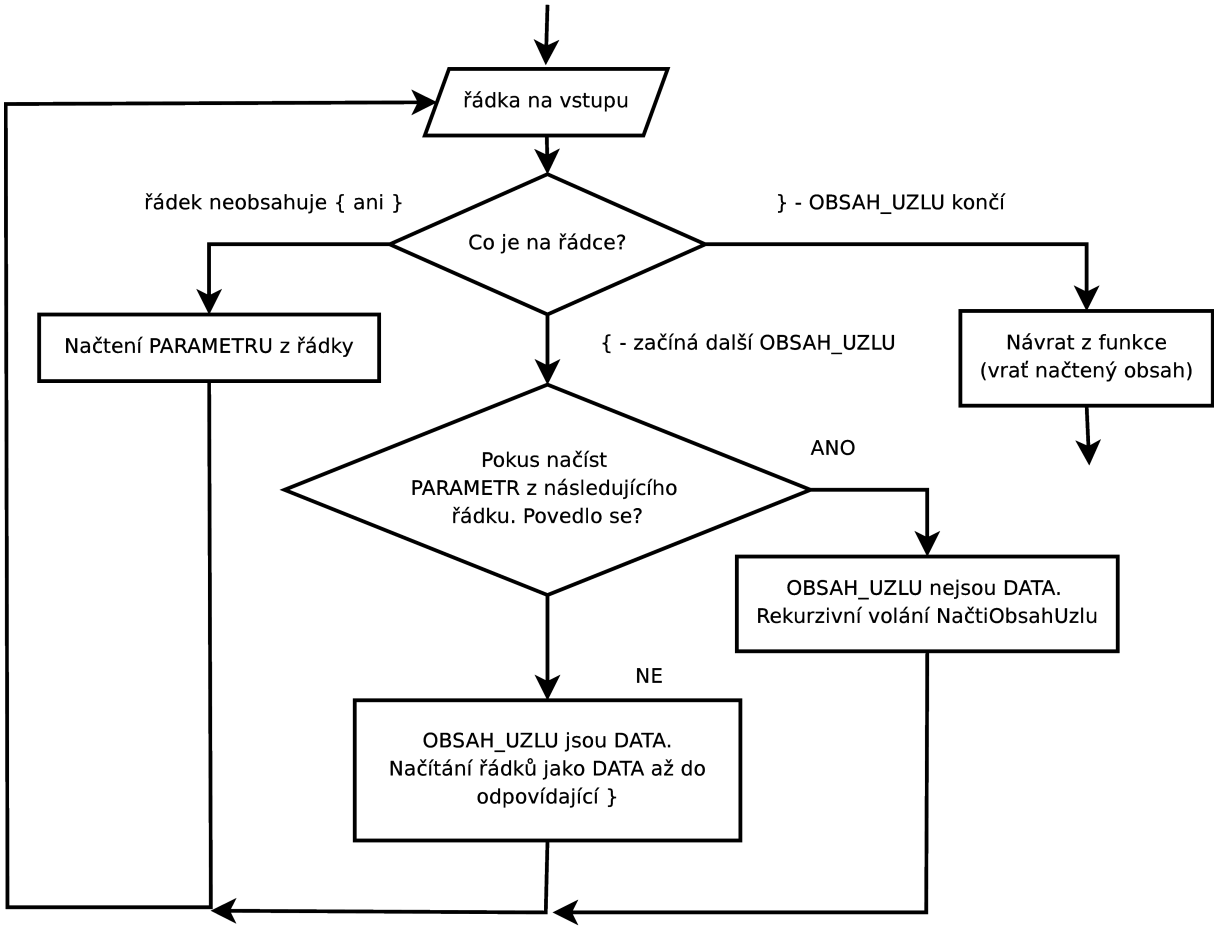
1 V některých situacích např. závisí tvar stromu FHS uzlů na jménu tagu.

7.1.2 Postup při čtení a zápisu FHS

Obecně lze formát FHS popsat například takto[2]:

```
FHS         = OBSAH_UZLU
OBSAH_UZLU  = *( PARAMETR | { OBSAH_UZLU } ) | DATA
PARAMETR    = TAG HODNOTA
TAG         = řetězec velkých písmen a podtržítek
HODNOTA     = různá, závisí na tagu. Pokud je víceřádková,
              konec hodnoty určuje správné spárování
              závorek [] ()
DATA        = řetězec, který nelze interpretovat jako
              PARAMETR
```

Tato "gramatika" je použita v FHS knihovně při samotném čtení ze souboru (prvotní načítání).



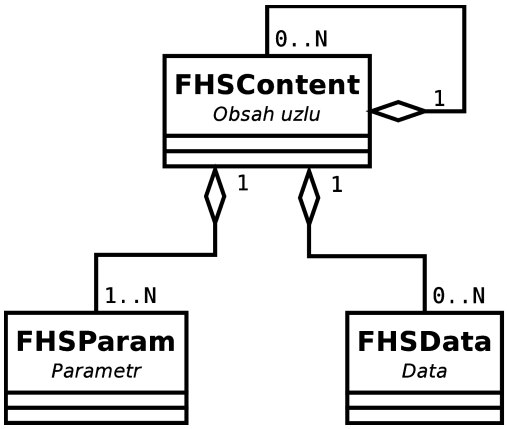
Obrázek 21: Vývojový diagram funkce NačtiObsahUzlu

Ve vývojovém diagramu chybí heuristiky, které sice využívají smyslu některých *tagů*, ale na druhou stranu zrychlují načítání. Například pro *tag* `GEOMETRY` lze předpokládat, že

2 Samozřejmě, že nevytvářím přesnou gramatiku. Například opomíjím konce řádek nebo odsazování tabelátory.

jeho OBSAH_UZLU jsou vždy data.

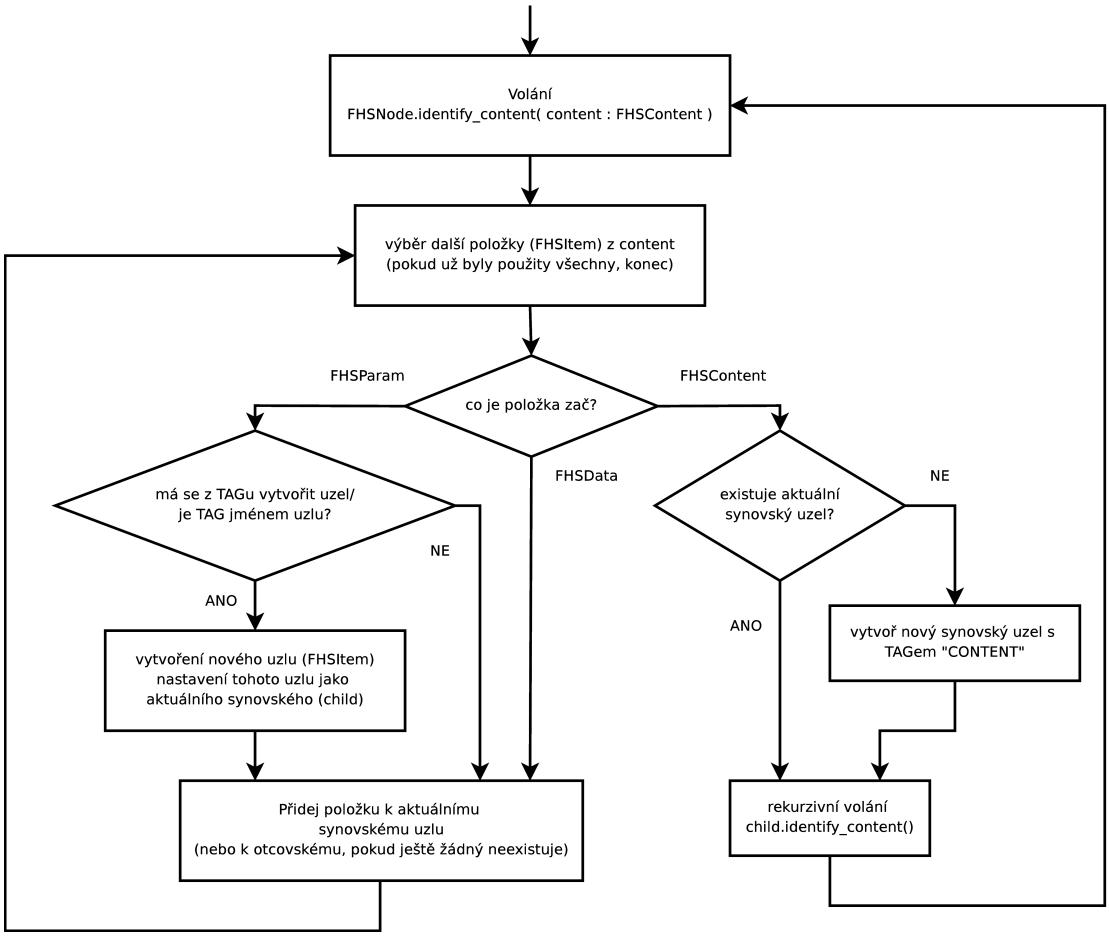
Data tvoří v souboru FHS velkou většinu. Rychlost funkce pro načítání dat výrazně ovlivňuje rychlost načítání celého souboru FHS. Proto je tato funkce přepsána do jazyka C.



Obrázek 22: Class reprezentace aktuálně načtené struktury

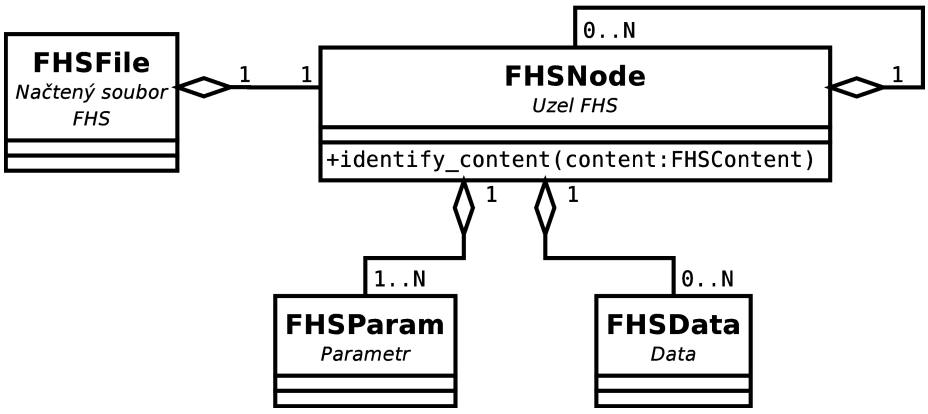
FHSParam, FHSData a FHSContent jsou sice odlišné třídy, jsou ale potomky třídy FHSItem a jsou uspořádány do jediného seznamu v pořadí odpovídajícímu pořadí v původním souboru.

Pro další zpracování knihovna využívá jména *tagů*, které určují začátky *uzlů*. Jde zejména o *tagy* MODEL, ASSEMBLY, GEOMETRY . Výsledkem je stromová struktura *uzlů*, jejich *parametrů* a *obsahu* (synovské *uzly* a *data*). *Tag* samotného *uzlu* je uchován jako speciální případ *parametru.*



Obrázek 23: Vývojový diagram dalšího zpracování

Pozn.: *tag* `CONTENT` se nikde ve specifikaci FHS nevyskytuje. Knihovna ho používá pro označení míst v FHS souboru, kterým "nerozumí" (resp. není potřeba aby tyto části dokázala interpretovat). Při budoucím ukládání se tato struktura uloží tak, jak byla uložena původně.



Obrázek 24: Class diagram načteného FHS (pro velký počet nejsou zobrazeny všechny položky a metody)

S touto strukturou pracuje aplikace VR Aid téměř neustále a z dosavadních zkušeností plně vyhovuje svému účelu.

7.1.3 Knihovna FHS

Knihovna FHS je navržena s ohledem na velké FHS soubory a s tím související nedostatek paměti. Například třída FHSData rovnou počítá s tím, že data nejsou v paměti, ale jsou uložena na disku. Instance FHSData obsahuje jen pozici v původním souboru a délku dat. Instance FHSData má proto malou velikost, lze ji často kopírovat a bez větších omezení s ní pracovat.

K práci s velkým množstvím dat dojde až při ukládání - data se v této fázi přečtou z disku (ale najednou a rychle, jejich analýza dle specifikace FHS už není potřeba). Výsledkem je zrychlení odezvy během samotné práce s aplikací. Uživatel musí čekat pouze při načítání a ukládání.

7.2 formát VRP

Formát VRP se ve ŠKODA AUTO, a. s. používá v těsné spojitosti s FHS. Lze v něm definovat některé parametry scény, ale hlavně slouží k přepínání různých částí FHS stromu a k usnadnění tohoto přepínání přímo při samotné prezentaci. Přepínat, zobrazovat a skrývat lze buď větve FHS stromu (např. různá provedení části automobilu) nebo materiály, které jsou pro FHS uzly použity.

Aktuální stav zobrazených částí FHS stromu (variant) lze uložit do variant setu (a samozřejmě opět vyvolat). Variant sety jsou součástí souboru VRP.

Na rozdíl od FHS, k formátu VRP neexistuje dokumentace. Tato část obsahuje popis té části VRP, která je potřebná k práci s variantami a variant sety.

7.2.1 Popis formátu VRP

VRP používá jednoduchý řádkový zápis:

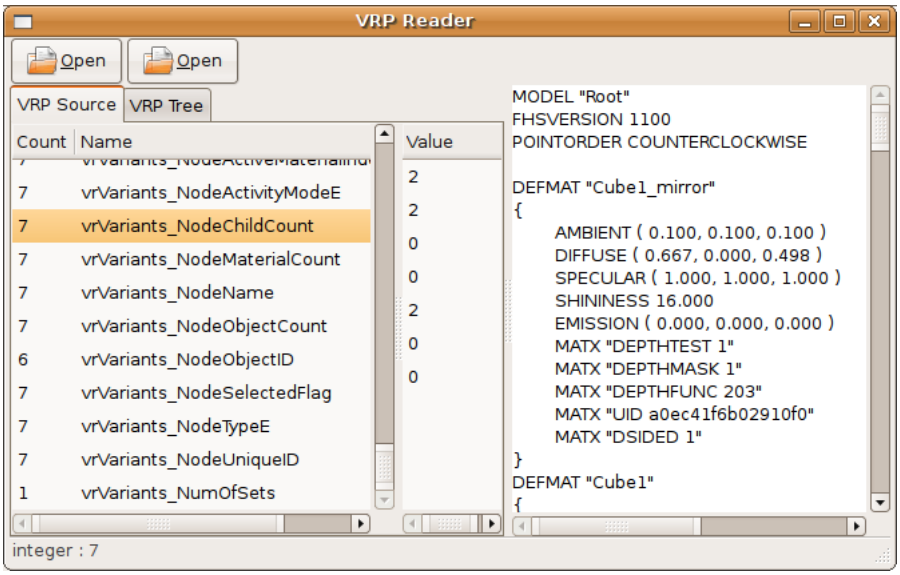
`KLÍČ : DATOVÝ_TYP : 0xB6HODNOTA0xB6`

`HODNOTA` je z obou stran ohraničena znakem `0xB6` . Hlubší smysl použití tohoto znaku nebyl zjištěn, ale knihovna VRP ho při zápisu zachovává kvůli kompatibilitě s VD2.

V souboru VRP se často vyskytuje více řádků se stejným `KLÍČEM` (více řádků, kde se shoduje `KLÍČ` i `DATOVÝ_TYP`, ale liší se `HODNOTOU`) . U těchto `KLÍČŮ` záleží na pořadí řádků a jev nich například uložena struktura VRP stromu.

7.2.2 Pomůcka vrp_reader

Pro účel zkoumání formátu VRP jsem napsal jednoduchý program vrp_reader, který dokáže poměrně přehledně zobrazit data VRP souboru.



Obrázek 25: vrp_reader

Program vrp_reader vytváří skupiny řádků podle KLÍČŮ a HODNOTY ukazuje v samostatném seznamu. Dokáže také analyzovat VRP strom a po načtení souboru FHS vyhledávat FHS uzly odkazované pomocí UID.

DATOVÝ_TYP

je celé číslo 1 - 5 a označuje typ `HODNOTY`

1 - boolean (zapsán jako 0 - false, 1 - true)
2 - integer
3 - float
4 - float
5 - string

jak se liší `DATOVÝ_TYP` 3 a 4 se nepodařilo zjistit a knihovna VRP je nerozlišuje. Jsou však zachovány a při zápisu jsou použity původní hodnoty (kvůli kompatibilitě s VD2).

7.2.3 Podrobný popis VRP

Naprostá většina VRP KLÍČŮ není pro popis variant FHS důležitá. Definují rozmístění světel, omezení scény a další parametry, pro které je vhodnější využít přímo VD2. Zde jsou popsány jen KLÍČE důležité pro strukturu VRP stromu a variant sety. U každého KLÍČE je popsáno, co znamená počet řádků s tímto KLÍČEM a jaký má význam pořadí řádků.

vrVariants_NodeName

počet řádků - počet VRP uzlů
pořadí řádků - pořadí VRP uzlů při procházení VRP stromu způsobem pre-order
DATOVÝ_TYP - string
HODNOTA - zobrazované jméno VRP uzlu

vrVariants_NodeUniqueID

počet řádků - počet VRP uzlů
pořadí řádků - pořadí VRP uzlů při procházení VRP stromu způsobem pre-order
DATOVÝ_TYP - string
HODNOTA - jedinečný identifikátor VRP uzlu

vrVariants_NodeSelectedFlag

počet řádků - počet VRP uzlů
pořadí řádků - pořadí VRP uzlů při procházení VRP stromu způsobem pre-order
DATOVÝ_TYP - boolean
HODNOTA - určuje, zda je uzel vybrán. VRP uzel je zobrazen pouze, pokud jsou vybráni všichni jeho předci.

Pokud je uzel VRP zobrazen, jsou zobrazeny všechny z něj odkazované FHS uzly.

vrVariants_NodeTypeE

počet řádků - počet VRP uzlů
pořadí řádků - pořadí VRP uzlů při procházení VRP stromu způsobem pre-order
DATOVÝ_TYP - integer
HODNOTA - 1 - jakákoli podmnožina potomků může být vybrána
2 - právě 1 potomek může být vybrán
3 - všichni potomci jsou vybráni

vrVariants_NodeChildCount

počet řádků - počet VRP uzlů
pořadí řádků - pořadí VRP uzlů při procházení VRP stromu způsobem pre-order
DATOVÝ_TYP - integer
HODNOTA - počet přímých potomků

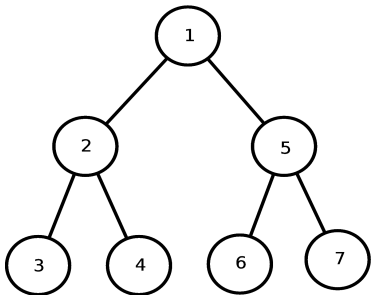
Tento KLÍČ určuje strukturu VRP stromu.

Příklad:

část VRP souboru

```
vrVariants_NodeChildCount:2: 2
vrVariants_NodeChildCount:2: 2
vrVariants_NodeChildCount:2: 0
vrVariants_NodeChildCount:2: 0
vrVariants_NodeChildCount:2: 2
vrVariants_NodeChildCount:2: 0
vrVariants_NodeChildCount:2: 0
```

odpovídající stromová struktura



vrVariants_NodeObjectCount

počet řádků - počet VRP uzlů
pořadí řádků - pořadí VRP uzlů při procházení VRP stromu způsobem pre-order
`DATOVÝ_TYP` - integer
`HODNOTA` - počet odkazovaných FHS uzlů

vrVariants_NodeObjectID

počet řádků - celkový počet odkazovaných FHS uzlů
pořadí řádků - pořadí VRP uzlů při procházení VRP stromu způsobem pre-order. Pro každý odkaz na FHS uzel 1 řádek.
`DATOVÝ_TYP` - string
`HODNOTA` - UID odkazovaného FHS uzlu

vrVariants_NodeObjectID a vrVariants_NodeObjectCount spolu jednoznačně definují odkazy z VRP uzlů na FHS uzly.

vrVariants_NodeMaterialCount

vždy 0

vrVariants_NodeActivityModeE

vždy 1

vrVariants_NodeActiveMaterialIndex

vždy -1

vrVariants_NodeActiveFlag

počet řádků - počet VRP uzlů
pořadí řádků - pořadí VRP uzlů při procházení VRP stromu způsobem pre-order
`DATOVÝ_TYP` - boolean
`HODNOTA` - určuje, zda je uzel zobrazen. VRP uzel je zobrazen pouze, pokud jsou vybráni všichni jeho předci pomocí vrVariants_NodeSelectedFlag.

Pokud je uzel VRP zobrazen, jsou zobrazeny všechny z něj odkazované FHS uzly.

vrVariants_NumOfSets

počet řádků - 1
`DATOVÝ_TYP` - integer
`HODNOTA` - počet variant setů

vrVariantSet_UniqueID

počet řádků - počet variant setů
pořadí řádků - pořadí variant setů při zobrazování v menu
`DATOVÝ_TYP` - string
`HODNOTA` - jedinečný identifikátor odkazovaného FHS uzlu

vrVariantSet_NumOfNodes

počet řádků - počet variant setů
pořadí řádků - pořadí variant setů při zobrazování v menu
`DATOVÝ_TYP` - integer
`HODNOTA` - počet VRP uzlů, které ovládá tento variant set

Všechny variant sety ovládají přibližně stejný počet uzlů a to plný počet uzlů ve VRP souboru. Jen pokud by na konci (dle pre-order) vrVariantSet_NodeActiveFlag byly nuly, pak se tyto VRP uzly do vrVariantSet_NumOfNodes nepočítají.

vrVariantSet_NodeID

počet řádků - součet všech vrVariantSet_NumOfNodes
pořadí řádků - pořadí VRP uzlů v jednotlivých variant setech
`DATOVÝ_TYP` - string
`HODNOTA` - ID VRP uzlu

vrVariantSet_NodeActiveFlag

počet řádků - součet všech vrVariantSet_NumOfNodes
pořadí řádků - pořadí VRP uzlů v jednotlivých variant setech
`DATOVÝ_TYP` - boolean
`HODNOTA` - určuje, zda je příslušný VRP uzel vybrán při aktivování variant setu

vrVariantSet_Name

počet řádků - počet variant setů
pořadí řádků - pořadí variant setů při zobrazování v menu.
`DATOVÝ_TYP` - string
`HODNOTA` - jméno odkazovaného FHS uzlu

vrVariantSet_ActiveMaterialIndex

vždy -1

7.2.4 Knihovna VRP

V knihovně VRP bylo třeba (stejně jako v případě FHS) udělat nějaké kompromisy. Například výchozí hodnoty pro nový soubor VRP nejsou uloženy ve zdrojovém kódu ani v konfiguračním souboru (těchto výchozích hodnot by muselo být přes 400 a jen by vytvářely zmatek). Místo toho se načte soubor default.vrp a následně se provedou změny `KLÍČŮ`, které jsou důležité (popsány výše).

Soubor default.vrp je možno také nahradit libovolným jiným VRP souborem (jeho původní stromová struktura a variant sety jsou ignorovány).

8 Uživatelská dokumentace

K ovládání programu VRAid je potřeba znát dohodu s VR-many a řídit se jejími pravidly. Tato dohoda je podrobně popsána v kapitole 3.2 .

Samotné ovládání programu VRAid je snadno pochopitelné z popisu implementace aplikace v kapitole 6 (zejména sekce o implementaci GUI 6.2 - 6.15).

9 Programátorská dokumentace

V této části jsou popsány všechny moduly programu VRAid a jejich funkce, třídy a metody.

9.1 fhs.py

Modul fhs.py obsahuje všechny prostředky nutné pro práci se samotným souborem FHS nebo FHSGZ. Z nestandardních modulů používá pouze modul fhsc.so .

9.1.1 důležité konstanty

NODE_NAMES

Konstanta obsahuje jména FHS tagů, ke kterým se knihovna fhs.py bude chovat jako k uzlům stromu FHS. K tagy, jejichž jména v seznamu NODE_NAMES nejsou, se chová jako k obecnému obsahu souboru FHS. Umí je jen přečíst ze souboru, uložit a změnit odsazení při změně vzdálenosti od kořene.

9.1.2 samostatné funkce

make_unique

Parametrem je seznam. Vrátí nový seznam, který obsahuje hodnoty ze seznamu, ale pouze jedinečné výskyty.

stack_brackets

Používá se při analýze víceřádkových parametrů. Na vstupu dostane řádku textu a seznam otevřených závorek (které byly otevřeny na začátku řádky). Vrací nový seznam závorek, které jsou otevřené na konci řádky.

filename_is_FHS

Detekuje podle koncovky souboru, zda je soubor (na vstupu je jméno) typu FHS.

filename_is_FHSGZ

Detekuje podle koncovky souboru, zda je soubor (na vstupu jméno) typu FHZGZ. FHSGZ je zagzipovaná verze formátu FHS.

get_keyword

Na vstupu dostane řádku a snaží se na ní najít klíčové slovo. Klíčová slova (tagy) v FHS mohou obsahovat pouze velká písmena a podtržítko. Vrací nalezené klíčové slovo nebo None.

9.1.3 třída FHSReader

Třída abstrahuje čtení ze souboru po řádkách. K tomu obsahuje užitečné funkce, které poskytují informace o stavu načítání souboru ve formátu FHS. Například udržuje řádkovou pozici při čtení ze souboru.

__init__

Instance třídy FHSReader je inicializována se jménem souboru, ze kterého se bude číst.

open

Inicializuje proměnné instance pro začátek souboru; nastaví čítač řádek na 0, apod. Otevře soubor.

close

Zavře spravovaný soubor.

get_line

Vrátí řádku textu, která je na aktuální řádkové pozici. Může vrátit i jakoukoli následující řádku (zajistí přečtení ze souboru). Řádky před aktuální pozicí nejsou potřeba a jsou zahazovány.

get_position

Vrací pozici souboru, která odpovídá začátku aktuálního řádku. Může vrátit pozici i pro následující řádky (ale ne pro předešlé).

get_total_size

Vrátí velikost spravovaného souboru v bytech.

segment_end

Detekuje, zda je aktuální řádek koncem segmentu. Segment v FHS je část mezi odpovídajícími složenými závorkami.

segment_start

Detekuje, zda aktuální řádek je začátkem segmentu.

arbitrary_data_follows

Detekuje, zda jsme na začátku segmentu a zda uvnitř segmentu jsou jen obecná data a žádné FHS uzly. K tomu využívá buď to, že data FHS mají specifickou strukturu, nebo že FHS uzel GEOMETRY už další uzly neobsahuje (jen data).

read_data_c

Předpokládá, že aktuální pozice je na začátku segmentu a přečte všechna data až do konce tohoto segmentu. K tomu využívá funkci napsanou v C. Posune aktuální řádek za přečtená data. Vrátí začátek přečtených dat a pozici dat v FHS souboru. Při požadavku na celá data je tak možné přímo načíst data z FHS souboru bez nutnosti analyzovat FHS soubor znovu.

read_data_python

Chová se stejně jako read_data_c, ale nepoužívá C. Používá se při čtení z FHSGZ, kdy není možné jednoduše číst data v C. Používá se také na platformě windows, kde není snadné

zkompilovat funkci pro Python napsanou v C.

next_line

Posune aktuální pozici o jeden řádek dopředu.

multiline_param_start

Detekuje, zda na aktuálním řádku začíná víceřádkový parametr. Například transformační matice jsou v FHS zadávány na více řádkách. Používá funkci stack_brackets a zjišťuje, zda jsou na konci řádku otevřené nějaké závorky.

read_multiline_param

Přečte všechny řádky víceřádkového parametru a vrátí tyto řádky. Posune aktuální pozici na konec parametru.

read_line

Přečte a vrátí jeden řádek a posune aktuální pozici.

get_line_position

Vrátí aktuální řádkovou pozici.

9.1.4 třída FHSGZReader(FHSReader)

Potomek třídy FHSReader. Pracuje stejně, ale se soubory typu FHSGZ.

open

Přečte všechna data z disku, rozzipuje je a uchová v operační paměti.

get_total_size

Vrací velikost rozzipovaných dat v bytech.

9.1.5 třída FHSItem

Předek všech prvků formátu FHS. Definuje virtuální funkce, které musejí všichni potomci implementovat.

save

Všichni potomci FHSItem se musejí umět uložit do otevřeného souboru. Popis této funkce je u potomků FHSItem velmi podobný, proto je uveden jen zde. Metoda save dostane jako parametr otevřený soubor, do kterého uloží údaje z instance. Stará se přitom zejména o správné odsazení, na které je program VD2 citlivý.

copy

Všichni potomci FHSItem musí umět vytvořit svou vlastní kopii.

9.1.6 třída FHSParam(FHSItem)

Třída představuje jeden parametr. Umí pracovat se jménem parametru (tagem) a formátovat řádky parametru pro různé účely (zobrazení, uložení do souboru, analýzu hodnoty).

get_formated_lines

Vrátí řádky parametru vhodné pro zobrazení v GUI.

get_keyword

Vrátí jméno (tag) parametru.

get_value_string

Vrátí hodnotu tagu jako řetězec. Odstraní případné uvozovky.

get_value

Vrátí hodnotu parametru - odstraní tag.

9.1.7 třída FHSParamMirror(FHSParam)

Vytvořením instance vznikne vždy parametr představující zrcadlení (podle roviny souměrnosti automobilu).

9.1.8 třída FHSData(FHSItem)

Třída představuje obecná nerozpoznávaná data. Udržuje záznam o tom, z jakého souboru byla data přečtena, z jakého offsetu a jak byla data dlouhá. Samotná instance FHSData je tak velmi malá, je možné s ní rychle pracovat a objekt mnohokrát kopírovat. Dále instance udržuje první řádek (nebo část prvního řádku, pokud je příliš dlouhý).

get_formated_lines

Vrací první řádek pro náhledové zobrazení FHS dat.

save

Přečte data z originálního souboru (spoléhá na to, že nebyl změněn) a po řádkách ukládá do otevřeného souboru. Přitom upravuje odsazování od začátku řádky.

9.1.9 třída FHSContent(FHSItem)

Instance FHSContent představuje obsah v segmentu FHS (mezi párem složených závorek).

from_reader

FHSContent se umí přečíst. Používá předaný objekt FHSReader a vytváří si strukturu parametrů (FHSParam), dat (FHSData) a dalšího obsahu (FHSContent).

get_all_data

Vyhledá a vrátí seznam všech datových uzlů (rekurzivně).

9.1.10 třída FHSNodeReplacementInfo

Třída se používá pro evidenci, které díly se mají zaměnit za které. Je úzce spojená s aplikací VRAid a nemá žádné univerzální použití.

9.1.11 třída FHSNode(FHSItem)

Představuje uzel FHS stromu, kterému knihovna fhs.py rozumí. Musí mít tedy jméno z fhs.NODE_NAMES . Udržuje záznamy o svých parametrech, odkazy na potomky a rodiče. Udržuje zvlášť ID a UID. ID uzlu je jedinečný identifikátor v rámci knihovny fhs.py . UID je jednoznačný identifikátor v rámci souboru FHS. Uzel nemusí mít nutně UID.

getId

Metoda třídy. Generuje ID, které ještě tato třída nevygenerovala (nemusí být nutně jedinečné v FHS stromu).

getUID

Jako parametr dostane seznam UID a vygeneruje takové UID, které ještě v seznamu není. Používá se pro generování nových jedinečných UID v rámci FHS stromu.

subnodes

Vrátí přímé potomky - FHS uzly.

all_subnodes

Vrátí všechny potomky - FHS uzly.

sibling_nodes

Vrátí FHS uzly na stejné úrovni.

pure_name

Jméno uzlu je řetězcová hodnota prvního řádku prvního parametru. Všechny uzly musejí mít alespoň jeden parametr (ten, který má tag uzlu).

pure_name_extension

Vrací příponu jména uzlu; řetězec za posledním podtržítkem ve jménu. Pokud podtržítko ve jménu není vrací None.

pure_name_without_extension

Vrací jméno uzlu bez přípony.

copy

Zkopíruje FHS uzel, ale negeneruje nové ID ani UID.

from_reader

FHSNode se umí načíst. Napřed vytvoří FHSContent a pak tento objekt analyzuje. Prochází parametry a zjišťuje, které parametry představují začátky dalších FHS uzlů.

identify_name

Uloží si jméno získané z prvního parametru jako položku. Rekurzivně zavolá na svoje potomky - FHS uzly.

get_params

Vrátí skutečné parametry, tedy všechny kromě prvního, který představuje jméno uzlu.

identify_uid

Vyhledá parametr, který obsahuje UID a uloží si UID jako položku. Rekurzivně zavolá na svoje potomky - FHS uzly.

register_id

Jako parametr dostane asociativní pole mapující ID uzlu na odkazy na příslušný uzel. FHS uzel přidá sám sebe do tohoto asociativního pole.

reassign_uid

Využívá se při kopírování části FHS stromu, uzel zkontroluje, zda jeho UID je jedinečné (pomocí předaného asociativního pole). Pokud není vygeneruje si nové a zaznamená, jak změnil své UID (zapíše to do dalšího předaného asociativního pole). Záznam o změněných UID se později použije pro úpravu variant.

addAssemblyChild

Přidá nad sebe další FHS uzel ASSEMBLY. Používá se v okamžiku, kdy je potřeba uzel GEOMETRY obalit uzlem ASSEMBLY. V FHS stromu by se neměly vyskytovat díly jako uzly GEOMETRY. Díl by měl mít tag ASSEMBLY a obsahovat další uzel GEOMETRY.

save_model

Předpokládá, že tento FHS uzel představuje tag MODEL (kořen FHS stromu) a uloží ho. Uzel s tagem MODEL je potřeba ukládat trochu jinak než ostatní FHS uzly.

save_params

FHS uzel uloží svoje parametry do otevřeného souboru.

save_content

FHS uzel uloží svoje potomky do otevřeného souboru.

get_UID_param

Vyhledá parametr obsahující informaci o UID.

get_params_by_keyword

Vyhledá všechny parametry FHS uzlu s konkrétním tagem.

delete_params

Smaže parametry FHS uzlu, jejichž tagy odpovídají předanému řetězci.

store_UID_in_params

UID se od načtení uzlu mohlo změnit. Tato metoda změní parametry tak, aby UID odpovídalo.

store_name_in_params

Jméno uzlu se od načtení mohlo změnit. Tato metoda změní parametry tak, aby jméno odpovídalo.

check_uid

Zkontroluje, zda UID uzlu je v předaném seznamu. Pokud je, vygeneruje se nové a

přidá se do seznamu UID.

check_id

Stejně jako check_uid, ale s ID.

get_all_parents

Vrátí seznam všech rodičů (až ke kořeni)

get_all_data

Vrátí všechny datové uzly v celém podstromě.

get_all_sources

Datový uzel obsahuje informaci o souboru, ze kterého by načten. Metoda vrátí všechna taková jména souborů v podstromu.

9.1.12 třída FHSFile

Třída, která udržuje informace o FHS souboru. Obsahuje kořenový FHS uzel (root), mapování FHS uzlů podle ID (nodeByID) a mapování FHS uzlů podle UID (nodeByUID).

load

Jako parametr dostane jméno souboru, typ souboru (FHS nebo FHSGZ) a objekt, kterému se má hlásit průběh načítání. Načte FHS ze souboru a přitom hlásí fázi načítání. Hlásí to vždy po načtení nějakého datového uzlu (z principu je obtížné hlásit průběh načítání v rámci jednoho datového uzlu).

notify_data_loaded

Spočítá průběh načítání podle velikosti souboru a aktuální pozice. Uvědomí objekt, kterému se má průběh načítání hlásit.

registerNode

Přidá údaje jednoho FHS uzlu do nodeByID a nodeByUID.

registerReplacement

Zaregistruje údaje o plánované záměně uzlu. FHS uzel který má být vyměněn je tak vyhledatelný na základě jména uzlu, kterým má být nahrazen. To urychluje proces záměny dílů za nové.

register_nodes

Zaregistruje všechny FHS uzly (uzly v podstromu uzlu root).

get_all_nodes

Vrátí seznam všech zaregistrovaných uzlů.

save

Stejně jako load. Průběh ukládání se hlásí po každém uložení datového uzlu.

copy

Vytvoří kopii celé struktury FHSFile. Struktura není příliš velká, protože neobsahuje

geometrická data, ale jen odkazy na tato data.

update_node

Metoda pro výměnu dat FHS uzlu dílu. Jako parametr dostane FHSNode, kde se budou měnit data, a FHSFile, která obsahuje nová data. Funkce vezme první uzel GEOMETRY, který najde v fhs_file, vezme jeho data a nahradí jimi data v FHS uzlu. Nahradí pouze data, zachová původní materiál, transformační matici, apod.

9.2 fhsc.c

Modul obsahuje drobnou funkci napsanou v C. Tato funkce má ale významný vliv na rychlost načítání souboru FHS. Pokud je místo této funkce použita funkce napsaná v Pythonu, rychlost načítání se sníží 3x (i tak ale bude přibližně stejná jako rychlost načítání souborů FHS v programu VD2).

Funkce fhsc dostane jako parametr otevřený soubor a pozici, na které začíná datový segment. Čte soubor dokud nenarazí na odpovídající konec segmentu (přečte tedy i vnořené segmenty). Vrátí délku datového segmentu v bytech a pozici začátku řádky za koncem datového segmentu.

9.3 fhs_display.py

Modul fhs_display propojuje FHS strom a nějaký způsob zobrazení tohoto stromu. Třídy v modulu fhs_display slouží k ovládání několika různých zobrazení FHS stromu. Zobrazení pro identifikaci stávají dílů, identifikaci nových dílů, zadání toho, které díly se mají nahrazovat, nebo jen základní zobrazení FHS stromu. Každý z těchto způsobů zobrazení má specifické nároky a slouží k zadávání jiného typu dat.

Každé zobrazení uchovává jednak instanci FHSFile a jednak instanci gtk.TreeStore, který slouží k ukládání dat zobrazovaných jako stromová struktura. Zobrazování FHS stromu tedy není transparentní, ale dochází ke kopírování dat mezi instancemi fhs.FHSFile do gtk.TreeStore.

9.3.1 třída FHSDisplay

Třída FHSDisplay sdružuje metody společné všem ostatním zobrazením FHS stromů.

saveModelValues

Uloží jeden sloupec hodnot z gtk.TreeStore stranou, do zvláštního asociativního pole.

loadModelValues

Opět nahraje hodnoty do gtk.TreeStore (tím způsobí jejich zobrazení). Metody saveModelValues a loadModel values se používají když je potřeba zachovat hodnoty již zadané uživatelem, ale přitom nějak pozměnit FHS strom.

onAllNodesToggled

Zobrazuje (kopíruje z FHS stromu do gtk.TreeStore) jen důležité uzly, nebo všechny uzly. K tomu používá předefinovatelnou metodu nodeImportant.

nodeImportant

Metoda dostává jako parametr ukazatel do stromu gtk.TreeStore . Vyhledá příslušný FHS uzel a rozhoduje, zda je či není důležitý. Jak se určuje důležitost uzlu je popsáno v

kapitole 6.7.1 .

markNodes

Přidá do výběru uzly, jejichž UID dostane jako parametr.

newTreeStore

Každé zobrazení FHS stromu spravuje jiná data a potřebuje jiné sloupce v gtk.TreeStore. Tato metoda je určena k předefinování.

setFHSFile

Dostane jako parametr instanci FHSFile a zajistí její zobrazení.

nodeToStoreTuple

Každé zobrazení FHS stromu spravuje jiná data a potřebuje jiné sloupce v gtk.TreeStore. Tomu musejí odpovídat i data, která se do gtk.TreeStore ukládají. Tato metoda převede FHS uzel na data, která se přidají do gtk.TreeStore. Tato metoda je určena k předefinování (v potomcích FHSDisplay) tím je možné udržovat kód pro kopírování FHS uzlů na jednom místě.

textToStoreTuple

Platí totéž jako pro nodeToStoreTuple.

showAllNodes

Vrací True pokud jsou zobrazeny všechny FHS uzly a nejen ty důležité, jinak False.

findFirstIter

Hledá první uzel gtk.TreeStore s nějakou konkrétní hodnotou. Jako parametr dostane hodnotu, která se hledá a index sloupce, ve kterém se má hodnota hledat.

loadFHSNode

Zajistí, aby z FHSFile do gtk.TreeStore byl převeden jeden konkrétní uzel. Jinak by bylo potřeba znovu zkopírovat všechny uzly.

onTreeViewButtonPress

Metoda která obsluhuje akce uživatele v gtk.TreeView . V případě FHSDisplay zařídí zobrazení parametrů FHS uzlu (při kliknutí na řádek představující FHS uzel).

getSelectedNode

Zjistí jaký řádek gtk.TreeStore je zrovna vybrán a vyhledá odpovídající FHS uzel.

saveExpansion, loadExpansion

Uloží stav zobrazení stromu, tj. které uzly jsou rozbalené (a jsou zobrazení i potomci). Umožňuje obnovit předchozí stav rozbalenosti stromu. Využívá se, když uživatel přechází mezi různými dialogy. Když se vrátí, strom je rozbalený tak, jak ho uživatel zanechal. To mu šetří práci.

9.3.2 třída FHSDisplayAddNodes(FHSDisplay)

Třída specializovaná na zadávání uzlů skupin. Důvod a okolnosti zadávání uzlů skupin

jsou popsány v kapitole 6.2 .

getNodesToBeAdded

Vrátí seznam uzlů (FHSNode), které byly označeny jako uzly skupiny.

onTreeViewButtonPress

Při dvojkliku je uzel vybrán jako uzel skupiny. Pokud jsou nějaké uzly v podstromu označeny jako uzly skupiny, jejich označení se zruší.

9.3.3 třída FHSDisplaySelectRoot(FHSDisplay)

Zobrazení FHS stromu, které slouží k zadání kořene pro vytvoření nového VRP (viz kapitola 6.2).

onTreeViewButtonPress

Dvojklikem je označen uzel, který bude sloužit jako kořen pro strom VRP. Označení ostatních uzlů (bylo-li jaké) je zrušeno.

9.3.4 třída FHSDisplayDetectUsedParts(FHSDisplay)

Zobrazení FHS stromu, které slouží zadávání parametrů dílů.

onTreeViewButtonPress

Dvojkliknutím na díl se mění označení z nepoužitelného na použitelný a naopak.

part_info

GTK umožňuje uložit data, která se nají zobrazovat i jinam než do gtk.TreeStore. Programátor pak musí poskytnout funkci, která data pro zobrazení vrátí na základě řádku a sloupce, který se má zobrazit. V tomto případě part_info vyhledává parametry uzlu ve struktuře aktuální verze prezentace.

part_edited

Podobně jako v případě part_info, ale dochází k editování dat uživatelem.

expand_managed

Funkce pro pohodlí uživatele. Při prvním zobrazení je strom rozbalen tak, aby byly vidět všechny použitelné uzly (například ty automaticky identifikované viz 6.8) a nebyly rozbalené žádné další, které by jen rušily.

9.3.5 třída FHSDisplayCorespondence(FHSDisplay)

Zobrazení FHS stromu, které slouží k zadání toho, jaký nový díl nahradí jaký stávající.

onTreeViewButtonPress

Při pravém kliknutí na nový díl, kterým se má nahradit stávající se zobrazí popup menu. V popup menu je možné plán nahrazení zrušit.

replacementToStoreTuple

Podobně jako nodeToStoreTuple v fhs.FHSDisplay .

9.3.6 třída FHSDisplayTessellation(FHSDisplay)

Zobrazení FHS stromu, které slouží k zadání velikosti tessellace pro celé podstromy v FHS stromu. Při zjišťování nastavené tessellace FHS uzlu použije VRAid tessellaci nejbližšího předka, který ji nastavenou má.

onTreeViewButtonPress

Pravím kliknutím na políčko s tessellací je možné vybrat z popup menu tessellaci, nebo tessellaci vymazat. V popup menu je možno vybírat z několika základních tessellací a pak z tessellací, které jsou již v aktuální struktuře verze prezentace použity.

tessellation_edited

Kromě výběru z menu je možné také tessellaci přímo editovat.

9.4 generation_wizard.py

Modul obsluhuje posloupnost 2 dialogů, kterými uživatel ovládá skutečné generování nové verze prezentace.

9.4.1 třída GenerationWizardPage1

Uživatel může zadat jméno nové verze prezentace a nebo určit,že díly se budou zaměňovat v aktuální prezentaci.

load_parts

Přechod ke druhému dialogu.

9.4.2 třída GenerationWizardPage2

Nejprve se určí, jaké natessellované díly bude třeba načíst. U každého FHS uzlu VRAid zjistí, zda bude nahrazován (využije uložený plán - identified_replacements) a jakým dílem a s jakou tessellací. Podle toho sestaví cesty k souborům. Po jednom soubory načte pomocí FHSFile.

modify_presentation

Skutečně změní prezentaci (vytvoří novou strukturu verze prezentace, změní FHS i VRP). Pokud se některé soubory nepovedlo načíst z disku, je upozorněn uživatel. Pokud přesto chce prezentaci změnit, výměna dílů se v případě nenačtených souborů zruší.

9.4.3 třída GenerationWizard

Třída řídí průchod mezi dialogy. Po úspěšném ukončení generování vezme nově vytvořenou strukturu verze prezentace a přidá ji do struktury správy prezentace.

9.5 helpers.py

9.5.1 samostatné funkce

Pomocné funkce pro zobrazování jednoduchých dialogů; hlášení o chybě, odsouhlasení, vstup jednoduché hodnoty.

9.6 presentation.py

Modul obsahuje třídy pro práci se strukturami popsanými v kapitole 6.16 .

9.6.1 třída PresentationStructure

Třída představuje strukturu verze prezentace. V budoucnu by bylo vhodné třídu přejmenovat tak, aby odpovídala použitým pojmům.

json_equivalent

Použití API knihovny demjson.py . Vrací instanci jako asociativní pole se jmény položek jako klíče a hodnotami položek jako hodnoty.

copy

Zkopíruje celou strukturu verze prezentace.

update

Jako parametr dostane jinou verzi prezentace. Změní svoje hodnoty tak, aby odpovídaly předaným hodnotám.

9.6.2 třída Presentations

Třída představuje strukturu (správy) prezentace. V budoucnu by bylo vhodné třídu přejmenovat tak, aby odpovídala použitým pojmům.

get_names

Vrátí všechna jména verzí prezentace.

addPresentation

Přidá další strukturu verze prezentace.

getCurrent

Vrátí strukturu verze prezentace, která je nastavena jako aktuální.

get_pres_by_name

Vrátí strukturu verze prezentace na základě jména verze. Vzhledem k omezenému počtu verzí nepoužívá VRAid k vyhledávání struktury verze prezentace asociativní pole, ale prochází všechny struktury verzí v seznamu.

updateExistingParts

Vytvoří seznam jedinečných jmen stávajících dílů ve všech strukturách verzí prezentace a uloží ho jako položku.

save

Uloží celou strukturu správy prezentace do souboru. Může dostat jako parametr otevřený soubor nebo jméno souboru. Pokud nedostane ani jedno, použije jméno, ze kterého byla struktura původně načtena (položka v instanci Presentations)

load

Nahraje strukturu správy prezentace ze souboru, přitom použije knihovnu demjson.

Postupně bere asociativní pole představující objekty a vytváří instance inicializované hodnotami z příslušného asociativního pole.

push_filename, pop_filename

Používá se při ukládání do dočasného souboru. Umožňuje na okamžik změnit jméno souboru a pak se vrátit k původnímu.

baseDir

Z celé cesty k souboru vr_aid vytvoří cestu do adresáře prezentace.

get_presentation_nodes_fhs

Jako parametr dostane jméno verze prezentace a instanci FHSFile. Vrátí všechny uzly FHS, které patří do této verze.

get_presentation_nodes_variant

Jako parametr dostane jméno verze prezentace a instanci Variants. Vrátí všechny uzly variant, které patří do této verze.

9.7 presentation_manager.py

Modul, který řídí jednotlivé dialogy při identifikaci dílů, při vytváření plánu výměny dílů a při zadávání tessellace. Jednotlivé dialogy si při startu kopírují strukturu správy prezentací a při návratu z dialogu buď původní strukturu nahradí a nebo tu zkopírovanou zahodí (podle toho, zda dialog byl ukončen přijmutím nebo odmítnutím provedených změn).

9.7.1 samostatné funkce

analyzePartString

Funkce dostane jako parametr řetězec, který pokládá za jméno dílu. Pokusí se v něm identifikovat parametry pomocí regulárního výrazu. Používá se při automatické identifikaci stávajících i nových dílů.

detectPartString

Funkce dostane jako parametr řetězec a zjišťuje, zda se jedná o díl. To zjišťuje podle koncovky řetězce. Pokud se jedná o koncovku nějakého FHS, CSB nebo CAD souboru (s případnou další koncovkou .mirror nebo .original), pokládá řetězec za jméno dílu.

pureNameToFilename

Odstraňuje koncovku .original nebo .mirror ze jména dílu, čímž vytváří původní jméno souboru.

load_column_widths, get_column_widths

Používá se na různé instance gtk.TreeView. VRAid umožňuje uživateli nastavit šířku sloupců a jejich šířku zachovává v rámci jedné prezentace.

Tyto funkce jsou také využity ve třídě PM_CheckCorespondencePage, kde jsou použity 2 instance gtk.TreeView nad sebou, a je třeba udržovat u obou stejnou šířku sloupců.

cmp_date

Funkce dostane 2 řetězce, které pokládá za datum. Porovná datumy, které tyto řetězce

reprezentují. Vrací stejné hodnoty jako vestavěná funkce cmp.

cmp_version

Funkce dostane 2 řetězce, které pokládá za verze dílů. Porovná čísla verzí.

9.7.2 třída PM_MainPage

Dialog základního pohledu na variant sety, varianty, strom FHS a parametry uzlů FHS.

update

Dialog začne používat nové instance FHSFile, Variants a Presentations, které metoda dostane předané jako parametry.

accept

Zapíše aktuální šířky sloupců do struktury správy prezentace.

9.7.3 třída PM_DetectUsedPage

Dialog pro identifikaci stávajících dílů. Identifikaci dílů uživatelem obstarává třída FHSDisplayDetectUsedParts.

update

Tuto metodu je volá PresentationManager při spuštění dialogu. Provádí automatickou identifikaci stávajících dílů.

accept

Identifikované díly jsou přidány do struktury aktuání verze prezentace.

9.7.4 třída PM_DetectNewPage

Dialog pro identifikaci nových dílů na disku. Tato třída obsluhuje svoje GUI sama, protože funkčně nesouvisí se zobrazování stromu FHS.

update

Tuto metodu je volá PresentationManager při spuštění dialogu. Provádí automatickou identifikaci nových dílů.

sortMethod

Metoda určující, jak budou nové díly řazeny v seznamu, podle priority, jakou by jim měl uživatel přisuzovat. Nahoře jsou zobrazeny ještě neidentifikované díly (je třeba je identifikovat). Ty může uživatel identifikovat ručně nebo říci že nebudou použity. Pod nimi jsou zobrazeny použitelné díly a nejníže nepoužitelné díly.

onTreeViewButtonPress

Dvojklikem může uživatel přepnout díl na použitelný nebo nepoužitelný.

9.7.5 třída PM_CheckCorespondencePage

Dialog pro spárování nových identifikovaných použitelných dílů se stávajícími díly. Zobrazuje 2 pohledy. Jeden na FHS strom prezentace a druhý na nové identifikované a

použitelné díly. Udržuje si údaje o tom, kolikrát by měl být nový díl v generované prezentaci použit.

replacement_added, replacement_removed

Udržuje údaje o tom, kolikrát má být nový díl v generované prezentaci použit (resp. v připravovaném plánu generování).

update

Volá ji PresentationManager při spuštění dialogu. Automaticky vytváří počáteční plán záměny dílů. Pokud ke stávajícímu dílu existuje odpovídající nový díl, použije nejnovější verzi.

get_node_replacement_pairs

Vrátí nový plán záměny dílů. Plán je asociativní pole, klíče tvoří UID stávajících uzlů a hodnoty jména nových dílů.

9.7.6 třída PM_ChangeTessellationPage

Dialog pro nastavení plánované tessellace. Obsahuje pouze triviální metody. Samotné zadávání tessellací obstarává FHSDisplayTessellation.

9.7.7 třída PresentationManager

Tato třída řídí spouštění jednotlivých dialogů pro:

- identifikaci stávajících dílů
- identifikaci nových dílů
- vytváření plánu výměny dílů (identifikace nahrazovaných dílů)
- zadávání plánovaných tessellací
- generování nové verze prezentace

9.8 presentation_wizard.py

Modul průvodce vytvořením nové struktury správy prezentace.

9.8.1 třída PresentationWizardPage1

fhs_open, fhs_open_win

Načte soubor FHS z disku. Přitom podle možností použije funkci pro načítání dat v C a dialog pro zobrazení průběhu načítání souboru FHS.

add_presentation_existing

Na základě seznamu uzlů skupin automaticky vytvoří seznam jmen verzí prezentace a vytvoří novou strukturu správy prezentace.

add_presentation_new

Na základě seznamu uzlů skupin a jména první verze prezentace vytvoří pod uzly skupin uzly verzí.

9.8.2 třída PresentationWizardPage2

Dialog pro výber FHS uzlu, který bude sloužit jako kořenový uzel pro varianty. Uživatel musí zvolit takový uzel, který je předkem všech uzlů skupin.

create_variants

Vytvoří strom variant a jeden variant set, který zobrazí (zatím) jedinou verzi prezentace.

9.8.3 třída PresentationWizard

Třída řídí přechody mezi stránkami průvodce vytvořením nové prezentace (první přechod od VD2 k VRAid).

9.9 progress.py

9.9.1 třída ProgressDialog

Jednoduchý dialog zobrazující průběh načítání souboru. K zobrazování průběhu načítání souboru je třeba podpora programování vláken. Ta bohužel ve spojení windows a GTK nefunguje a tak VRAid ve windows nezobrazuje průběh načítání souboru.

9.10 variant.py

Třídy v modulu variants.py datově odpovídají třídám v souboru VRP. Obsahují však navíc užitečné funkce pro práci se stromem variant a s variant sety.

Původně měl být formát VRP pouze jedním z formátů pro zaznamenání variant. Modul variants byl vytvořen kvůli oddělení zápisu do souboru (mělo být více možností) od práce s variantami (ta měla být společná).

9.10.1 třída VariantNode

Datově odpovídá třídě vrp.VRPNode.

name_extension

Vrací příponu ve jménu - řetězec znaků za posledním podtržítkem ve jménu (name). Pokud ve jménu podtržítko není, vrací None.

name_without_extension

Vrací část jména před příponou (včetně podtržítka)

all_subnodes

Vrací všechny (přímé i nepřímé) potomky.

all_parents

Vrací seznam všech předků až ke kořeni.

all_selected_subnodes

Vrací všechny potomky, kteří jsou zobrazeni (jsou vybráni a všichni jejich předci také)

reassign_uid

Používá se při kopírování části FHS stromu a odpovídající části stromu variant. Zjistí se staré a nové UID FHS uzlů a tyto UID se musí změnit i při kopírování ve stromu variant.

9.10.2 třída Variants

Datově odpovídá třídě VRPTree.

get_new_id

Generování nových jedinečných identifikátorů pro instance VariantNode a VariantSet.

copy_node

Zkopíruje uzel varianty (vygeneruje nové id). Přidání uzlu varianty do vyhledávacích struktur (nodes_by_id) se řeší zvlášť.

register_node

Zaregistrování uzlu varianty do vyhledávacích struktur,

copy

Bezpečné (s vygenerováním nových id) zkopírování celého objektu Variants.

activate_set

Zapnutí variant setu. Vyberou se jen ty varianty, které patří do příslušného variant setu (předán jako parametr).

9.10.3 třída VariantSet

Datově odpovídá třídě VRPTree.

9.10.4 třída VariantToVRPAdapter

Třída, která řeší konverzi mezi (VariantNode, VariantSet, Variants) a (VRPNode, VRPSet, VRPTree).

__init__

Při inicializaci dostane instance alespoň jeden instanci Variants nebo VRPTree. Může dostat i obě, ale při spuštění konverze bude ta původní zničena (zahozena).

extractVariantFromVRP, storeVariantToVRP

Převede datovou strukturu VRPTree do odpovídající struktury Variants (resp. naopak). Datově si obě struktury odpovídají, takže algoritmus konverze je triviální.

9.11 variant_display.py

9.11.1 třída VariantsDisplay

Řídí zobrazování variant a variant setů. Použita v dialogu PM_MainPage.

9.12 vrp.py

Formát VRP používá plochý jmenný prostor. Stromová struktura variant je v něm vyjádřena sémanticky (viz kapitola 7.2). Modul VRP obsahuje nejnutnější funkce pro převod plochého VRP na stromovou strukturu.

9.12.1 třída VRPNode

Neobsahuje žádné metody představuje jen strukturu uzlu pro vytváření stromu variant. Funkcionalitu při ukládání a nahrávání obstarává třída VRPTree.

popis položek

položka	typ	popis	odpovídající VRP tag
name	string	obsahuje jméno VRP uzlu. Tento text se zobrazuje při zobrazování VRP stromu.	vrVariants_ NodeName
selected	0 nebo 1	Určuje, zda je tento VRP uzel vybrán.	vrVariants_ NodeSelectedFlag
type	1, 2 nebo 3	Určuje, jaké varianty mohou být vybrány. 1 - jakákoli podmnožina children 2 - právě jeden z children je vybrán 3 - všechny children jsou vždy vybrány	vrVariants_ NodeTypeE
id	string	jednoznačný identifikátor VRP uzlu ve stromě variant	vrVariants_ NodeUniqueID
objects	seznam stringů	seznam UID těch FHS uzlů, které ovládá tento VRP uzel. FHS uzly jsou zobrazeny, právě když je vybrán tento VRP uzel a všichni jeho předci.	vrVariants_ NodeObjectID
children	seznam instancí VRPNode	seznam přímých potomků. Neodpovídá přímo žádnému VRP tagu. Získává se ze sémantiky VRP souboru.	
child_count	celé číslo	délka seznamu children. Po načtení VRP souboru child_count odpovídá seznamu children. Při ukládání se tato hodnota znovu spočítá dle seznamu children.	
original_id	boolean	Udává, zda id tohoto VRP uzlu bylo původně načteno z VRP souboru. Pokud nebylo mohou být ID mohou být při ukládání přegenerována.	

9.12.2 třída VRPSet

Neobsahuje žádné metody, jen strukturu pro variant set.

popis položek

položka	typ	popis	odpovídající VRP tag
name	string	jméno variant setu - zobrazuje se v editoru	vrVariantSet_ Name
id	string	jednoznačný identifikátor variant setu	vrVariantSet_ UniqueID
original_id	False nebo True	Udává, zda id tohoto variant setu bylo původně načteno z VRP souboru. Nová ID mohou být při ukládání přegenerována.	
selected_variants	seznam instancí VRPNode	seznam variant, které mají být vybrány při aktivaci tohoto variant setu	vrVariantSet_ NodeActiveFlag

9.12.3 třída VRPTree

Třída představující VRP soubor. Po načtení se souboru je možno přímo pracovat s jednotlivými VRP uzly (variantami) a variant sety. Je možné přímo měnit vybrané uzly ve variant setu - přidáním nebo odstraněním instance VRPNode z VRPSet.selected_variants. Změna stromu variant se provádí přímou změnou instancí VRPNode - přidáním nebo odstraněním instancí VRPNode z VRPNode.children. Není potřeba měnit VRPNode.children_count (to se spočítá automaticky při ukládání). Lze měnit odkazované FHS uzly - přímým přidáním nebo odebráním řetězců UID z/do seznamu VRPNode.objects.

Při přidávání a odebírání VRP uzlů a variant setů je třeba ručně vygenerovat nové id (odlišné od ostatních), je třeba přidat nový uzel (resp. variant set) do nodes a nodesByID (resp. sets a setsByID). Je možné také tuto práci nechat na modulu variants.py .

položka	typ	popis
root	VRPNode	kořen stromu variant
nodesByID	asociativní pole	mapování instancí VRPNode podle jejich id
setsByID	asociativní pole	mapování instancí VRPSet podle jejich id

load

Jako parametr dostane cestu k souboru VRP. Načte data ze souboru do mezistruktury, tu následně analyzuje a vytvoří strom variant, příp. několik variant setů.

save

Podobně jako load

9.12.4 mezistruktura VRP

Soubor VRP je plochý a je tvořen samostatnými řádky ve tvaru:

název tagu;typ hodnoty;hodnota

Mezistruktura pouze uchovává tyto řádky jako slovník seznamů a zachovává pořadí, tj. pořadí hodnot v seznamu odpovídá pořadí řádků se stejným názvem tagu.

loadStructure, saveStructure

Nahraje a uloží mezistrukturu do souboru.

9.13 vr_aid.py

9.13.1 třída VRAid

Hlavní třída aplikace VRAid. Řeší hlavně načítání a ukládání souborů a řeší chyby (nepovolený přístup do adresáře, zda se mají při ukládání přepsat existující soubory, a pod.)

10 Další možnosti vývoje

Aplikace VR Aid dokáže výrazně urychlit práci na prezentacích na Pracovišti virtuální techniky. Nejedná se však o dokonalý nástroj. V této kapitole jsou uvedena některá vylepšení, která by pomohla použitelnosti nástroje.

10.1 Práce s FHB

Aplikace VR Aid dokáže nyní pracovat pouze s formáty FHS a FHSGZ. Formát FHB je binární a neexistuje k němu dokumentace. Implementace práce s tímto formátem by dále urychlila načítání a ukládání, pokládám ji však za velmi problematickou.

10.2 Propojení s Operou a DeltaGenem

V aktuální verzi program VR Aid musí VR-mani stále konvertovat CAD modely do FHS souborů sami a umisťovat tyto FHS soubory do adresářové struktury tessellací. Jejich práci by více usnadnilo, kdyby programy Opera a DeltaGen uměla použít přímo aplikace VR Aid.

Další možností je vytvořit distribuovaný systém pro konvertování CAD modelů Operou. Program DeltaGen již funkci distribuovaného konvertování (v rámci lokální sítě) sám o sobě obsahuje.

10.3 Undo / Redo

Funkce Undo a Redo by zlepšila ovládání aplikace. Veškerý stav aplikace je udržován ve struktuře správce prezentací, která není příliš velká. Pro implementaci by stačilo tuto strukturu ve vhodných okamžicích kopírovat a ukládat do seznamu pro Undo.

10.4 Generování kusovníku v XLS

Při vytváření nové verze prezentace musí VR-man také vytvořit kusovník dílů které jsou ve verzi prezentace použity. Tento kusovník sestavuje ručně a řídí se jednoduchými pravidly. Aplikace VRAid tak při vytváření nové verze má již k dispozici všechny informace, které VR-man používá při vytváření kusovníku. Bylo by tedy relativně jednoduché doplnit funkcionalitu, která by zajišťovala i automatické generování kusovníku.

11 Zhodnocení splnění cílů

Při vývoji programu VR Aid bylo mojí hlavní prioritou usnadnění práci zaměstnanců pracoviště virtuální techniky. S ohledem na tento hlavní cíl byly v průběhu vývoje učiněny některé změny v zadání oproti oficiálnímu zadání diplomové práce.

11.1 Hodnocení programu VRAid

Program je první svého druhu, neexistuje jiný podobný projekt, kde bych se býval mohl inspirovat. Jediné, z čeho jsem mohl při tvorbě programu vycházet byla dokumentace FHS a zkušenosti VR-manů. Oproti původnímu oficiálnímu zadání do programu přibyla schopnost pracovat se soubory VRP (přidružených k FHS). Nutno zmínit, že k formátu VRP neexistovala vůbec žádná dokumentace.

Oproti původnímu zadání VR Aid není schopen komunikovat s databází KVS. Je to způsobeno nevysvětlitelnými průtahy ze strany administrace KVS při žádostech o povolení přístupu do této databáze. Německá strana nikdy nevydala ani záporné stanovisko. Kvůli nemožnosti komunikovat s KVS bylo již zbytečné, abych do programu zapracovával funkcionalitu cachování stahovaných dílů a správu přístupových práv uživatelů (VR-manů)

Program VR Aid v současné verzi není schopen sám automaticky spouštět program Opera (ani DeltaGen), díky dohodě s VR-many je však možné výsledky konvertování používat.

Program VR Aid je tedy nakonec samostatnou lokální aplikací, která provází VR-mana při tvorbě nových verzí prezentací. Splňuje hlavní cíl projektu - usnadnit práci zaměstnanců pracoviště virtuální techniky a být pomůckou doplňující program VD2. Přínos programu VR Aid (zrychlení práce VR-mana) je zřetelný hlavně při větším počtu uzlů skupin (např. rozsáhlejší animace drobných součástí).

11.2 Další osud programu VRAid

Program VR Aid byl vyvíjen tak, aby doplňoval program VD2. V poslední fázi vývoje VR Aid však začalo pracoviště virtuální techniky postupně přecházet na program DeltaGen a s ním spojený formát CSB (binární, proprietární a bez dokumentace). S nasazováním programu DeltaGen se bude postupně snižovat intenzita používání programu VD2 (jehož vývoj se zastavil) a s tím i programu VR Aid.

Pracoviště virtuální techniky bude i nadále používat program VD2 při přípravách prezentací pro systém virtuální reality CAVE, protože systém CAVE není schopen pracovat se soubory CSB (DeltaGen). V této oblasti použití VD2 bude VRAid i nadále přínosem.

11.3 Další výsledky diplomové práce

Během práce na programu VRAid jsem vytvořil počáteční dokumentace k formátu VRP. Tato dokumentaci a knihovny pro práci s formáty FHS a VRP mohou být užitečné při dalším vývoji aplikací pro pracoviště virtuální techniky.

Při návrhu programu VRAid jsem shledal systém Messages mimořádně užitečným. Mě i Ing. Červenému výrazně zjednodušil komunikaci. Na druhou stranu musím přiznat, že jsem nezkoušel použít žádné jiné aplikace pro komunikaci programátora se zákazníkem.

12 Seznam použité literatury

[1] Robert Bankwitz : HyperKVS Dokumentation kvsclient - Technische Schnittstellenbeschreibung, 27. Februar 2004

[2] Jörg Merkl : Dokumentation Opera 2.0 - Datenaufbereitung für die Virtuelle Realität, 15.10.2004

13 Používané pojmy

CATIA	nejčastěji (ve Škodě Auto, a. s.) používaný formát CAD souborů
CSB	binární formát programu DeltaGen
FHS	textový formát pro popis virtuální scény
FHB, FHS.GZ	binární verze formátu FHS
generování prezentace	automatická výměna dílů v prezentaci za nové nebo za díly z jinou tessellací, prováděná podle plánu předem vytvořeného VR-manem
IGES	ANSI standard pro konstrukční data
JSON	JavaScript Object Notation, viz. 5.4
Opera	program pro konverzi různých formátů CAD dat do formátu FHS
VRP	textový formát přidružený k formátu FHS, umožňuje definovat varianty scény a přepínat mezi nimi
VR-man	pracovník virtuální techniky ve společnosti Škoda Auto, a. s.
VD2	Virtual Designer 2, hlavní nástroj VR-manů při tvorbě prezentací

14 Seznam definic

15 Seznam obrázků