Celý tento text se týká generování Visual C++ kódu v Rose pomocí odpovídajícího pluginu. Normání C++ funguje podobně, ale ostatní jazyky jsou naprosto odlišné. V dalším textu se často odkazuji na Rose, přitom mám na mysli právě ono "Rose Visual C++".

1 Main Features of Rose Visual C++

object oriented analysis and design

- **Model Assistant**: přidání běžných VC konstruktů do modelu, včetně MFC
- **Component Assignment**: vytváření komponent a přiřazování jazyků třídám
- **Round-trip engineering**: synchronizace modelu a kódu
- **Code generation**: generování a update kódu podle modelu, včetně **code patterns**
- **Reverse engineering**: vytvoření nebo update modelu podle kódu
- **COM support**: pomocí ATL a MIDL

2 Code Generation Mapping Rules

Motto firmy Rational: "I komentář je zdroják."

2.1 MDL Component = VC project

- nutný předpoklad pro generování kódu
  kód třídy se generuje až po přiřazení třídy do komponenty s jazykem VC++
- interface musí být přiřazen do IDL komponenty s jazykem VC++

2.2 Stereotype, Code Template, Model Properties

- ovlivňují to, jaký se vygeneruje kód (tělo)

2.3 COM object

- Rose generuje zvlášť IDL, CoClass a její implementaci v samostatné VC++ třídě.
- MIDL komponenta je obvykle vytvořena při generování kódu
  (stereotype=<<MIDL>>, type=MIDL)
- může být více MIDL komponent → více MIDL souborů
- editace pomocí Model Assistant

2.4 Documentaion

- vkládá se jako komentář vždy před prvek, ke kterému patří (třída, metoda, atribut)

2.5 CodeName

Pokud je to možné, jména tříd v C++ odpovídají jménům v Rose (class name + name.h + name.cpp), stejně tak jména operací, atributů a typů (typy=třídy).
CodeName mapuje jména z modelu do jmen kódu (při použití neanglických jazyků apod.) – podporuje: třídy, metody a proměnné (ve všech možných místech výskytu)
CodeName se zapíná v **Tools→Visual C++→Properties** (default: off).

2.5.1 Considerations

Pole jsou v Rose součástí názvu, ne typů: **$$$[10]:int** musí mít CodeName **cost[10]**, ne pouze **cost**.
Jména .h a .cpp souborů jsou u každé třídy generována podle CodeName.

Třídy z balíku MFC by neměly používat CodeName.
Obsah **code patterns** a **stereotypes** (např. friend) se nepřekládají.
Rose C++ a Rose Visual C++ se liší v práci s "$ expressions".
Nastavování a změna CodeNames se provádí v Model Assistantu.

2.5.2 Name Conversion

2.5.2.1 Vypnuté CodeName

Rose vyhodí neplatné znaky: z **@customer@** vytvoří `customer`.
Změna ilegálních jmen se provede i přímo v modelu!
Pokud v názvu nezbude ani znak, Rose oznámí chybu.

2.5.2.2 Zapnuté CodeName

Rose validuje pouze CodeName a generovaný kód, zatímco model nemění. Model se mění pouze při generování kódu pro nepojmenované role, které jsou automaticky pojmenovány.
Pokud element nemá CodeName, použije se přímo jméno z modelu, ale nevyhazují se neplatné znaky! Při pokusu vytvořit **@customer@** Rose oznámí chybu.

2.5.3 Neplatné znaky

Při správném používání CodeName otázka neplatných znaků vůbec nenastane.
Bez CodeName provádí Rose konverzi, při round-trip je obousměrná a Rose musí umět rozeznat jednotlivé názvy mezi sebou. Někdy vznikne chaos → Rose vyrobí chybu v projektu.
Je velmi doporučeno ve jménech v modelu:

- nepoužívat mezery
- nepoužívat znaky operátorů C++

Doporučení se týká jen jmen, ne obvyklých konstrukcí v Rose, kde je často "<", ">" nebo ":"!!!

2.5.4 Jména typů

Jména typů jsou překládána pouze v případě, kdy jde o jména tříd definovaných v modelu.
Rose se snaží najít CodeName typu takto:

1. Hledá v modelu třídu stejného jména jako daný typ.
2. Pokud nenajde, hledá třídu, která má CodeName stejné jako daný typ.
3. Při updatování modelu podle projektu se prohledávají nejprve třídy realizované stejnou komponentou, potom se hledá v celém modelu.

Pozn.: Rose nezakazuje duplicitní jména v modelu a/nebo CodeName, pouze při takovém stavu přidá hlášku do logu.

3 Logical View to C++ Mapping

Logical view = classes, operations, attributes, relationships, packages

3.1 Classes

3.1.1 Generovaný kód

Třída je generována podle stereotypu (viz kapitola 3.2).
Každý element třídy je public, private nebo protected.
Rose pro každou třídu generuje:

- #include odkazy na potřebné datové typy = jiné třídy
- deklarace třídy (podle jména, typu a předků (dědění))
- datové členy (atributy a vztahy s jinými třídami)
- deklarace členských metod + definice prázdných těl těchto metod
  (pro metody vracející hodnotu se generuje `return 0;`)
- dokumentace v podobě poznámek v kódu u každého elementu

- otravné absurdní nesmysly (oficiálně nazýváno `//##ModelID=123`)

3.2 Stereotypes

- stereotyp určuje, jaký se bude generovat kód
- Rose rozeznává: struct, union, enum, typedef, interface
- pokud je neuveden nebo je neznámý, generují se standardní .h + .cpp soubory
- stereotypy lze skládat, oddělují se čárkou nebo středníkem

3.3 Class Utilities

Class utility je zcela **statická třída** (všechny metody a atributy jsou statické).
Rose z toho generuje „C++ modul".
Generovaný kód je stejný jako u běžné třídy, ale u každého elementu je `static`.
Pozn.: Statickým atributům jsou také generovány definice v .cpp souboru.

3.4 Operace (členské metody tříd)

3.4.1 Stereotypy operací

- bez stereotypu: `result name(params);`
- abstract: `virtual result name(params)=0;`
- const: `virtual result name(params)const;`
- friend: `friend result fname(params);`
- static: `static result name(params);` - Rational tvrdí, že lokální proměnné takovéto metody jsou zachovány mezi voláními – to je podle mě blbost
- virtual: `virtual result name(params);`

3.4.2 Operation Semantics

Operation semantics (viz Operation Properties) se při generování C++ ignorují.

3.4.3 Parameter Passing

Lze klasicky definovat defaultní hodnoty parametrů: `result func(param:int=6);`

3.4.4 Get & Set Functions

Přístupové metody `Get_name` a `Set_name` se generují podle nastavení (viz Attribute Properties).

3.5 Proměnné objektů tříd

Vše se generuje podle očekávání, takže uvádím jen pár poznámek:

- scalar:int
- vector[100]:int
- pointer:int*
- reference:int&

3.6 Asociace mezi třídami

Rose generuje další proměnné podle "navigable association relationships" = čáry mezi třídami v diagramu.
nepojmenované asociace → dostanou defaultní jméno a Rose přidá warning do logu
Tato defaultní jména jsou zanesena přímo do modelu, takže při příštím generování už vše proběhne hladce.
Násobným asociacím lze uvést i způsob implementace (31):

- +objekty[50]:typ

- +objekty:vector

Rose defaultně používá pointery pro asociace. Pokud uvedete referenci jako součást názvu, Rose generuje referenci. Př. +role = +*role = pointer, +&role = reference, +role[25] = pole, +**role = pointer na pointer

3.7 Agregace (vlastnění)

Rose generuje kód jako proměnnou vlastníka, která je typu vlastněné třídy. Opět pozor na to, že Rose generuje jen to, co je „navigable".

3.8 Dependency relationship (závislosti)

Pro závislosti (dodavetel-odběratel) se negenerují žádné proměnné, pouze odběratel má `#include "dodavatel.h"` v .cpp souboru a `class dodavatel;` v .h souboru. (Je to dobře?)

3.9 Generalizace (dědění)

superclass=předek, subclass=potomek (33) (viz kontrast s množinami: podtřída je větší než nadtřída)
Rose generuje `#include "předek.h"` a jméno předka přidá potomkovi do seznamu děděných tříd `class potomek:předek {};`.

3.10 Další poznámky

- Rose vždy generuje jen to, co je "navigable"
- Rose ignoruje ByValue a ByRef. Jediná možnost, jak určit realizaci je uvést ji přímo do jména role (viz 3.5).
- Rose ignoruje multiplicity of roles – to už je fakt hrůza!
- Násobné asociace lze implementovat pomocí MFC:
  1. QuickImport MFC classes
  2. Model Assistant lze použít pro přiřazení kolekcí z MFC k jednotlivým rolím
  Pozn.: Tohle irRational už fakt přehání!!! Viz C++ knihovnu std.
- Packages nic negenerují = irRational™®

4 Component View to C++ Mapping

Komponenta je .exe/.dll/.ocx/.tlb/.idl/.odl soubor a odpovídá vždy jednomu VC projektu (.dsp).
Stereotyp komponenty je exe nebo dll (funguje automaticky po přiřazení projektu komponentě pomocí Component Assignment).

5 Deplyment View to C++ Mapping

Rose nepodporuje generování podle Deployment View (se dalo čekat).

6 Visual C++ to Rose Mapping

Pokud jsou v projektu „conditional compiler directives", tak Rose generuje model jen podle těch, které jsou v danou chvíli aktivní. (irRational tak asi chtějí zcela znemožnit použití Rose v praxi).
Mapping rules jsou inverzní k těm pro generování kódu. Pokud máte i komentáře ve zdrojácích podle stylu Rose, jsou i ony zahrnuty do dokumentace v modelu.
Při použití MFC jsou MFC třídy vloženy do modelu a mají nastenou vlastnost Generate=False.

6.1 Agregace (vlastnění)

irRational se chluví také tím, že při round-trip generování jsou zachovány defaultní nastavení rolí. Tzn. pro agregační role bez uvedení typu (tzn klasika) se při reverzním generování opět vytvoří role bez uvedení typu. Tzn. místo `jméno:typ` svítí na obrazovce jen `jméno`. (Funguje to ale úplně stejně, tak už to neřešme.)

7 Three-Tiered Model

Rose tohle nepodporuje, protože trojvrstevný model je jen jedním z mnoha možných modelů.

8 Round-Trip Engineering

Round-Trip = děláte to pořád dokola, až do dokončení projektu nebo zblbnutí z Rose (podle toho, co nastane dříve)
Code Update tool: generuje a updatuje kód Visual C++ podle modelu
Model Update tool: generuje nebo updatuje model podle Visual C++
Model Assistant: pomocník pro zakládání nebo změnu tříd, přidávání členských metod a proměnných. Umožňuje také do tříd přidávat Windows message handlers a MFC overrides.

Nový projekt lze začít modelováním i reverzním generováním z existujícího kódu (Rational tvrdí: „The real world frequently disallows ideal practices.“)

Aby to fungovalo, pracujte v daný okamžik buď pouze na modelu nebo pouze na zdrojáku. Potom proveďte synchronizaci. Pokud budete rýpat do obojího zaráz, tak vám to Rose „opraví“.

8.1 Další poznámky

- round-trip step-by-step viz strana 51
- Přesun metod do jiné třídy proveďte ve zdrojáku. Pokud to provedete v modelu, tak přijdete o původní tělo. (Asi je to důležité, neboť Rational na to upozorňuje hned dvakrát.)
- Synchronizujte model se zdrojákem co nejčastěji.
  Pozn.: Dokud to nebude v jednom IDE, tak je to stejně irRational™® chaos.
- Dobrá metoda, jak „zničit krtka“, je měnit `//ModelID=xxx` čísla ve zdrojáku. (Viz poznámku u předchozího bodu.)

9 Generování kódu (59)

Kód třídy lze generovat až po jejím přiřazení nějaké komponentě, která má jazyk Visual C++.
Kód lze generovat buď najednou pro celý projekt, anebo podle potřeby po jednotlivých třídách.

10 Reverse Engineering (69)

Pozn.: Rational tomu taky říká arechologie. (Někdy to může být taky perverse engineering, záleží na tom, kdo je autorem toho výchozího zdrojáku.)

Visual C++ projekt odpovídá jedné komponentě v modelu.