VYSOKÉ UČENÍ TECNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Modelování HW designu v UGE

Bakalárska práca

2007 Ladislav Varga

Čestné prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením Ing. Aleša Smrčku. V zdrojoch som uviedol všetky literárne pramene a publikácie, z ktorých som čerpal.

_____

Ladislav Varga

23. januára 2007

Abstrakt

Predmetom tejto práce bolo navrhnúť zásuvný modul pre program Univerzálny grafický editor, ktorý uživateľom umožní vytvárať návrh hardware. Návrh HW architektúry začína spravidla kreslením blokových schém navrhovaného systému. Návrh je potom zapísaný v niektorom z jazykov, slúžiacich pre popis hardware. Keďže implementácia návrhu v týchto jazykoch má modulárnu štruktúru, tzn. podobnú štruktúre blokovej schémy HW návrhu, je možné uvažovať o preklade schémy, ktorú v programe nakreslí vývojár, do zdrojového kódu jazyka pre popis hardware. Táto myšlienka má za cieľ uľahčiť návrhárom HW prácu, ktorá môže byť zautomatizovaná. Navrhnutý modul poskytuje prostriedky pre vytváranie HW schém a komponent na rôznej úrovni abstrakcie a interaktívne týmito úrovňami prechádzať. Modul tiež implementuje preklad nakreslenej HW schémy do kódu jazyka VHDL.

Kľúčové slová

Univerzálny grafický editor, UGE, dynamicky linkované knižnice, zásuvný modul, graf, návrh hardware, kreslenie schém, HW komponenty, preklad schémy do kódu VHDL

Abstract

The goal of this thesis is to create a plugin for application Universal graphic editor, which will allow users to design a hardware architecture. Design of hardware architecture usualy starts with drawing of block diagrams of system which is being developed. Next step is to transcribe this drawn design into some hardware description language (HDL). Since structure of hardware design written in HDL is modular, i.e. similar to the structure of its block diagram, it's possible to translate block diagram of hardware design into HDL source code. The point of this idea is to get rid of designer's work on re-writing the block diagram into HDL language, as this can be automated. Designed plugin allows users to create block diagrams and new hardware components on different layers and switch between these layers interactively. Modul also implements the translation of drawn diagram into VHDL source code.

Key Words

Universal graphic editor, UGE, dynamic link libraries, plugin, graph, diagram, hardware design, diagram drawing, hardware components, diagram translation into VHDL source code

Obsah

1 Úvod

Prudký rozvoj výpočtovej techniky v posledných rokoch neustále zvyšuje nároky kladené na číslicové systémy, ktoré sú základným stavebným prvkom číslicových počítačov. Dnešný návrháry číslicových systémov musia držať krok s týmto trendom a ich práca je preto čoraz náročnejšia.

Zložité systémy sa nedajú navrhovať klasickými technikami, so zvyšujúcou sa komplexnosťou sa návrh tvorí na stále vyššej úrovni abstrakcie. Návrh číslicových systémov môže byť prevádzaný na rôznych úrovniach abstrakcie, od popisu chovania systému na najvyššej úrovni, cez popis štruktúry systému, až k samotnej implementácií na úrovni najnižšej. Chovanie systému môžme popísať napr. slovne, matematicky, časovým diagramom, stavovým automatom, zápisom algoritmu v programovacom jazyku, atď. Jeho štruktúru môžme vyjadriť blokovou schémou, tabuľkou a taktiež ju zapísať v programovacom jazyku. V praxi sa tieto prístupy vhodne kombinujú – návrhár hardware (ďalej len HW) tvorí algoritmus pre riešenie daného problému a zároveň vytvára schému systému zloženú z jednotlivých komponent.

Evolúcií návrhu číslicových systémov sa samozrejme prispôsobujú aj nástroje, ktoré sú na tento účel používané. Pre popis HW architektúry sa dnes používajú jazyky označované ako HDL (Hardware Descripton Language), z ktorých najznámejšie sú Verilog a VHDL (Very High Speed Integrated Circuit Hardware Description Language). Návrh HW architektúry, zapísaný v HDL jazyku, má modulárnu štruktúru – skladá sa z blokov, vzájomne poprepájaných signálmi, ktoré slúžia na prenos informácií medzi jednotlivými blokmi. Tieto bloky tvoria jednotlivé HW komponenty na rôznej úrovni abstrakcie (pr. komponenta „radič zbernice" predstavuje vysokú formu abstrakcie a komponenta „multiplexor" nízku formu abstrakcie nakoľko sa jedná o jeden zo základných logických obvodov). Každá komponenta má rozhranie, ktoré tvorí skupina vstupných a výstupných signálov. Zmeny úrovne vstupných signálov ovplyvňujú chovanie danej komponenty – na základe jej vnútorného stavu a logických hodnôt signálov na vstupe sa menia hodnoty výstupných signálov. Týmto spôsobom HDL jazyky analogicky popisujú to, čo obsahuje návrh číslicového systému – jeho štruktúru a chovanie.

V posledných rokoch sa dôležitou súčasťou návrhu hardware stáva formálna verifikácia, aj vďaka niektorým neslávne známym chybám, ktoré boli odhalené až po uvedení produktov na trh (napr. chyba delenia u procesorov Pentium® firmy Intel®). Verifikácia pomáha dokázať, že požadované vlastnosti systému zostanú zachované vo všetkých stavoch, do ktorých sa systém môže dostať. Jedným zo základných prístupov formálnej verifikácie je „model checking" (výraz nemá ustálený preklad). Model checking patrí medzi vysoko-úrovňové metódy formálnej verifikácie. Vysoko-úrovňová formálna verifikácia je postup, pri ktorom sú zo systému

abstrahované len jeho dôležité vlastnosti (tie, ktoré chceme verifikovať), čím sa značne zníži komplexnosť a časová náročnosť verifikácie. Princíp model checking-u spočíva vo vytvorení modelu verifikovaného systému a následnej automatickej kontrole, ktorú vykonáva verifikačný nástroj. Kontrola modelu spočíva v prehľadávaní konečného stavového priestoru systému a overovaní, či daný stav zodpovedá požadovanému chovaniu systému. Model, ktorý sa pri model checking-u používa, je najčastejšie vytvorený z návrhu HW architektúry alebo návrhu software. Existuje niekoľko nástojov používaných pre model checking napr. SPIN, SMV a jeho deriváty Cadence SMV a NuSMV. Tieto nástroje majú podobu programovacieho jazyka, v ktorom je zapísaný model verifikovaného systému. Tieto nástroje potom dokážu automaticky prevádzať verifikáciu systému spôsobom popísaným vyššie.

1.1 Cieľ práce

Návrh HW architektúry vzniká spravidla „na papiery", tzn. kreslením blokových schém, diagramov, atď. Ten je potom návrhárom zapísaný v konkrétnom HDL jazyku. „Papierový" návrh má zároveň aj dokumentačný význam, čo je u väčších projektov nevyhnutné, či už na nich pracujú skupiny vývojárov alebo len jednotlivci. Cieľom tejto práce je vytvoriť nástroj, ktorý bude návrhárom HW umožňovať vytvárať návrhy v podobe kreslenia blokových schém číslicových obvodov na osobných počítačoch.

Keďže implementácia návrhu v HDL jazyku má modulárnu štruktúru, tzn. podobnú štruktúre blokovej schémy HW návrhu, je možné uvažovať o preklade schémy, ktorú nakreslí vývojár do zdrojového kódu HDL jazyka. Táto myšlienka má za cieľ uľahčiť prácu návrhárom HW, najmä z hľadiska prepisovania štruktúry návrhu do daného jazyka, ktoré spočíva v deklarovaní komponent a ich rozhraní a v ich vzájomnom poprepájaní. Práca pokrýva aj túto problematiku. Ako cieľový jazyk pre preklad schémy bol vybraný jazyk VHDL pre jeho rozšírenosť na pôde FIT a v projektoch, na ktorých sa fakulta podiela.

Dôvod prečo sme vyššie popisovali spôsob verifikácie metódou model checking je, že sa v praxi používa pre formálnu verifikáciu HW návrhu. Pri zápise modelu do nástroja používaného pre formálnu verifikáciu sa podobne, ako pri zápise do HDL jazykov, využívajú hierarchické konštruckie. Jednotlivé bloky tejto hierarchie sú u modelu HW návrhu spravidla komponenty, ktoré návrh obsahuje. To znamená, že zo schémy HW návrhu je možné vygenerovať kostru modelu pre verifikačný nástroj, čo môže byť užitočné pri práci na formálnej verifikácií HW návrhu.

2 Univerzálny grafický editor

V tejto podkapitole sa zoznámime s programom Univerzálny grafický editor (UGE), ktorý poskytuje prostriedky pre kreslenie a prácu s obecným grafom (viz. definícia 2.1). Popíšeme si motiváciu vzniku tohto programu, jeho koncepciu, prostriedky, ktoré používa pre prácu s grafom a prostriedky, ktoré poskytuje pre vývoj zásuvných modulov.

Definícia 2.1*:* Graf je usporiadaná dvojica (V, E), kde V je množina prvkov – vrcholy (uzly) grafu a E je množina dvojích prvkov z V – hrany grafu.

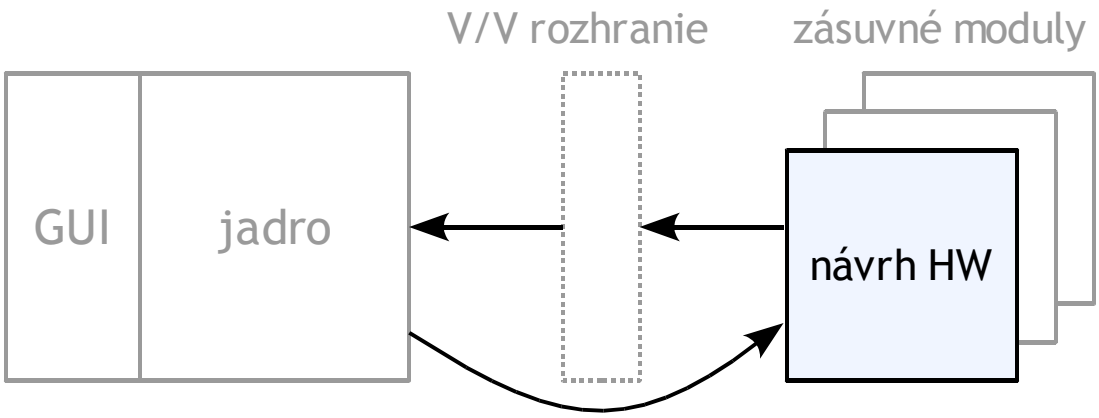
2.1 Motivácia

V praxi existuje veľké množstvo rôznych schém a diagramov, ktoré sú špeciálnym prípadom obecného grafu. V týchto schémach sa objavujú uzly (nech už vyjadrujú čokoľvek) a sú prepájané hranami, ktoré môžu popisovať určitý vzťah, postupnosť, prepojenie alebo prechody medzi uzlami. Keď vezmeme do úvahy len obor informatiky, ako príklad môžme uviesť schémy rôznych automatov (konečné, stavové, časované, ...), petriho sieťe, UML diagramy, ER diagramy, schémy databáz, schémy topológie sietí, vývojové diagramy atď. Pre kreslenie mnohých z týchto typov diagramov existujú nástroje, ktoré danú problematiku viac či menej komplexne pokrývajú. Všetky majú však spoločnú jednu vec – vo všetkých sa kreslia grafy. Prečo teda nevytvoriť jeden program, ktorý bude umožnovať kresliť rôzne typy grafov, diagramov a schém, ktoré majú spoločného predchodcu – obecný graf? Uživateľ by tak nemusel používať pre tvorbu troch odlišných typov diagramov tri rôzne programy, ktoré robia v podstate to isté len v inom „kabáte". V tomto momente by sa mohol objaviť argument, že nie je možné, aby jeden program pokryl tak široké spektrum heterogénnych grafov. Ale čo ak by program umožnoval jednoduchým spôsobom naprogramovať „podporu" pre nový typ grafu? Programátor by len definoval vzhľad prvkov pre daný typ diagramu/grafu, ich vlastnosti a takýmto spôsobom by vytvoril v programe podporu práve pre ten typ grafu, aký potrebuje.

2.2 Popis programu

Myšlienka Univerzálneho grafického editoru spočíva vo vytvorení prostredia, ktoré bude umožnovať prevádzať operácie s obecným grafom (vytváranie a úprava grafov, zmena vlastností grafových objektov, tlač, ukladanie, ...) koncovým uživateľom a zároveň bude poskytovať prostriedky k tomu, aby bol graf „programovateľný". tzn., aby bolo možné definovať vzhľad grafových objektov a ich parametre, pridávať objektom nové vlastnosti, ovplyvňovať chovanie grafu pri akciách uživateľa a pod. To umožní, že s grafmi a pomocou nich bude možné

prevádzať rôzne akcie od simulácie konečných automatov, cez monitorovanie rozličných systémov, až po generovanie schém zo získaných dát. Uplatnenie takého postredia je široké. Z týchto požiadavkov vychádza koncepcia programu znázornená na obrázku 2.1.



Obrázok 2.1: Schéma Univerzálneho grafického editoru.

Základ programu tvorí grafické jadro, ktoré obsahuje štruktúry uchovávajúce v pamäti inštanciu grafu, štruktúry grafických primitív a algoritmy pre prácu s grafickými primitívami. Prostredníctvom týchto základných prostriedkov vie jadro pracovať s obecným grafom. K jadru sa viaže grafické uživateľské rozhranie (GUI) pre zobrazenie výstupov jadra a interakciu s uživateľom. „Programovanie" vlastností a chovania grafu je možné pomocou zásuvných modulov (známych tiež pod názvom „plugin"). Pre komunikáciu medzi jadrom a zásuvnými modulmi slúži vstupno-výstupné rozhranie, ktoré je popísané v podkapitole 2.5.

2.3 Reprezentácia grafu

Logická štruktúra grafu v UGE sa skladá zo štyroch základných elementov:

- uzol
- hrana
- port
- skupina portov

Existujú 2 typu uzlov grafu:

- jednoduché – hrany sa pripájajú priamo k uzlu
- s portami – hrany sa pripájajú k portom, ktoré ležia na okraji uzlu

Existujú 4 typy hrán:

- priama – „úsečka" vedená medzi dvoma prípojnými bodmi (uzol alebo port)
- lomená – hrana môže byť zalomená v ľubovolnom uhle
- ortogonálna – zalomenia hrany sú pravouhlé
- oblá – hrana, ktorej grafická reprezentácia je beziérova krivka

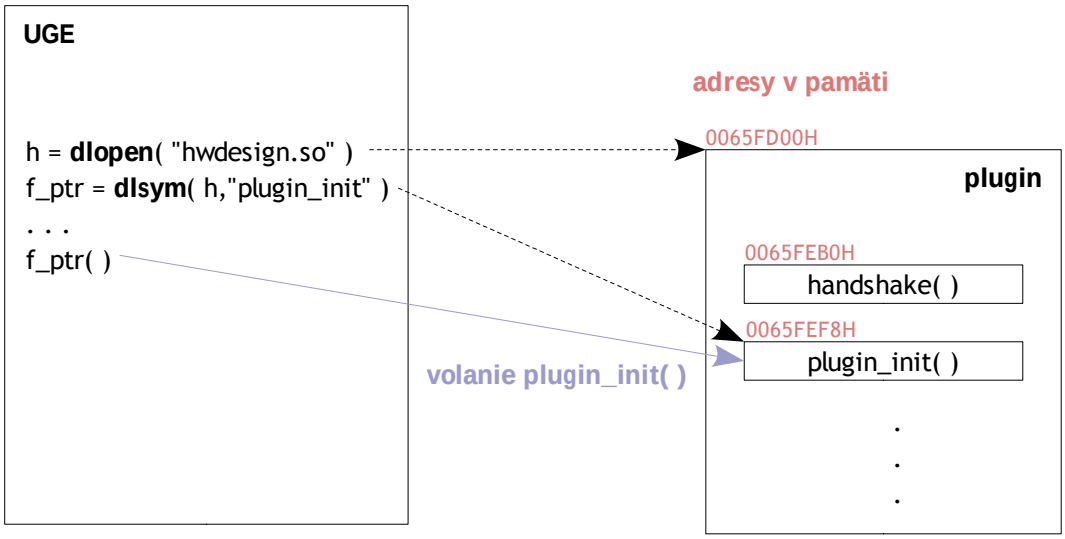
Porty môžu byť zoskupované do skupín portov. Atribúty portov potom určujú chovanie v rámci danej skupiny – porty môžu mať statickú pozícu voči uzlu alebo môžu byť rozmiestnené relatívne (rovnomerne) v rámci skupiny. Ďalší atribút portu určuje stranu uzlu, na ktorej je umiestnený (severná, južná, východná, západná) – to má samozrejme zmysel len u uzlov s grafickou reprezentáciu so štyrmi stranami. Každý objekt má v rámci grafu unikátny celočíselný identifikátor a unikátne meno (názov). Každý grafový objekt ma svoju grafickú reprezentáciu, ktorá je zložená z grafických primitív (elipsa, kruh, obdĺžnik, beziérova krivka, text, čiara). Uzol má vlastnosť, ktorá určuje o akú triedu uzlu sa jedná – každý modul si vytvára vlastné triedy uzlov a podľa tejto vlastnosti rozoznáva tie, ktoré mu „patria". Grafovým objektom je možné tiež pridávať nové vlastnosti základných datových typov (celé číslo, číslo v pohyblivej rádovej čiarke, reťazec, logická hodnota) „za behu". Toto sú základné prostriedky, ktoré jadro má pre prácu s grafmi. Štruktúra jadra UGE je samozrejme komplexnejšia a poskytuje ešte viac funkcií. Detailný popis je možné nájsť v práci [12], ktorá sa zaoberá návrhom a implementáciou jadra a jeho rozhrania.

2.4 Zásuvné moduly

Princíp zásuvných modulov spočíva v poskytovaní balíku funkcií cieľovému programu. Program, pre ktorý bol modul napísaný, môže tieto funckie volať a pracovať s hodnotami, ktoré mu funckie vrátia. Popri tom funkcie modulu môžu volať špecifickú skupinu funkcií programu a ovplyvňovať tak jeho činnosť. Tieto prostriedky zabezpečujú, že modul môže plne ovplyvňovať funkčnosť programu, ak je to potrebné.

Zásuvné moduly bývajú spravidla realizované ako dynamicky linkované knižnice. Princíp práce s dynamicky linkovanými knižnicami je vidieť na obrázku 2.2. Program si môže „otvoriť" knižnice za behu s využitím prostriedkov operačného systému a volať funkcie, ktoré knižnica obsahuje. „Otvorenie" knižnice sa prevádza volaním POSIX-ovej funkcie „dlopen", ktorá zabezpečí načítanie knižnice do operačnej pamäte a vráti ukazateľ na začiatok pamäte, kde knižnica leží. Ak chceme volať funkciu z dynamickej knižnice, musíme poznať jej meno. To predáme ako parameter ďalšej POSIX-ovej funckií „dlsym" spolu s ukazateľom na začiatok pamäte knižnice. Ak funkcia „dlsym" nájde v knižnici funkciu s daným názvom, vracia na ňu ukazateľ. Program, ktorý si knižnicu otvoril, tak môže pomocou ukazateľa túto funkciu

jednoducho zavolať.



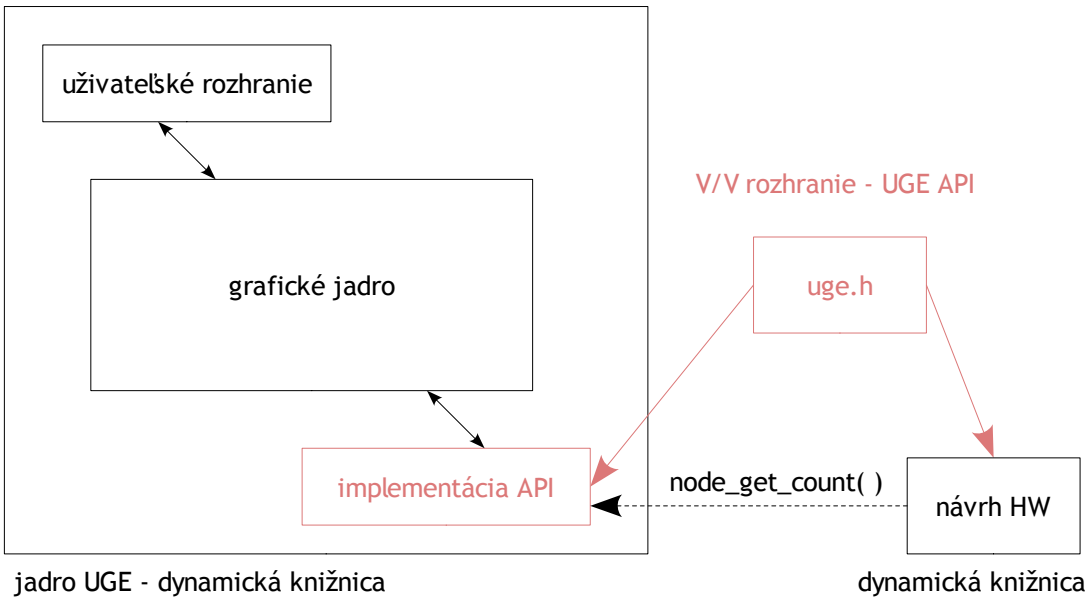
Obrázok 2.2: Volanie funkcií dynamických knižníc.

Spôsob komunikácie (rozhranie) medzi programom a zásuvným modulom obecne nie je pevne daný a je na návrhárovi programu, ako ho realizuje. Komunikácia medzi UGE a jeho zásuvnými modulmi prebieha pomocou prostriedkov vstupno-výstupného rozhrania UGE, ktoré je detailnejšie popísané v podkapitole 2.5. Priebeh komunikácie je nasledovný - hlavný program si pri spustení načíta všetky zásuvné moduly a z každého z nich zavolá funkciu, ktorou sa modul identifikuje (dá jadru vetieť svoj názov, verziu a popis). Keď uživateľ zvolí, že chce daný modul používať, je zavolaná funckia pre inicializáciu modulu, v ktorej si modul napojí funkcie pre spätné volanie na príslušné signály jadra. Funkcie pre spätné volania sú funkcie modulu, ktoré volá jadro (nepozná ich meno a volá ich pomocou ukazateľa). Signály generuje jadro na základe akcií, ktoré sú vyvolané (uživateľom programu alebo automaticky). Pri každom vygenerovanom signále jadro prehľadáva tabuľku signálov a volá postupne funckie, ktoré sú na daný signál napojené (medzi nimi aj funckie modulov). Takýmto spôsobom sú moduly informované o akciách prebiehajúcich v jadre a môžu tak reagovať napr. na pridávanie nových uzlov, zmenu ich vlastností, vyberatie položky menu atď. Na zasahovanie do činnosti jadra používajú moduly funckie vstupno-výstupného rozhrania, popísaného v nasledujúcej podkapitole.

2.5 Vstupno-výstupné rozhranie

Pre komunikáciu medzi samotným programom UGE a zásuvnými modulmi bolo navrhnuté vlastné aplikačné rozhranie (API) programu. Toto vstupno-výstupné rozhranie poskytuje modulom funkcie, pomocou ktorých môžu komunikovať s jadrom – získavať ľubovolné vlastnosti grafových objektov, zobrazovať užívateľom dialogové okná a pod. Funkcie API sú napísané v jazyku C, čo znamená vysokú univerzálnosť rozhrania. Modul môže byť tým pádom napísaný v ľubovoľnom jazyku, ktorý umožňuje volanie funkcií deklarovaných v jazyku C. Programátor modulu tak nieje viazaný na konkrétny programovací jazyk a môže použiť ten, ktorý mu najviac vyhovuje.

Moduly majú prístup k funkciám API prostredníctvom hlavičkového súboru, ako je vidieť na obrázku 2.3.



Obrázok 2.3: Schéma vstupno-výstupného rozhrania (API).

Schéma na obrázku 2.3 zobrazuje princíp prepojenia a komunikáciu medzi hlavným programom a zásuvným modulom. Funckie API sú definované v jadre UGE a ich prototypy sú deklarované v hlavičkovom súbore. Po jeho vložení môže plugin volať funkcie API – na obrázku 2.3 je ako príklad znázornené volanie funkcie *node_get_count*.

V nasledujúcom úseku kapitoly sú vymenované časti vstupno-výstupného rozhrania a

konkrétny popis jeho funkcií, ktoré sú vývojárovy zásuvného modulu k dispozícií pre manipuláciu s grafom, grafovými elementami a ich vlastnosťami. Modul pre návrh HW architektúry, ktorého návrh je predmetom tejto práce, využíva podskupinu z nich.

Funkcie V/V rozhrania UGE

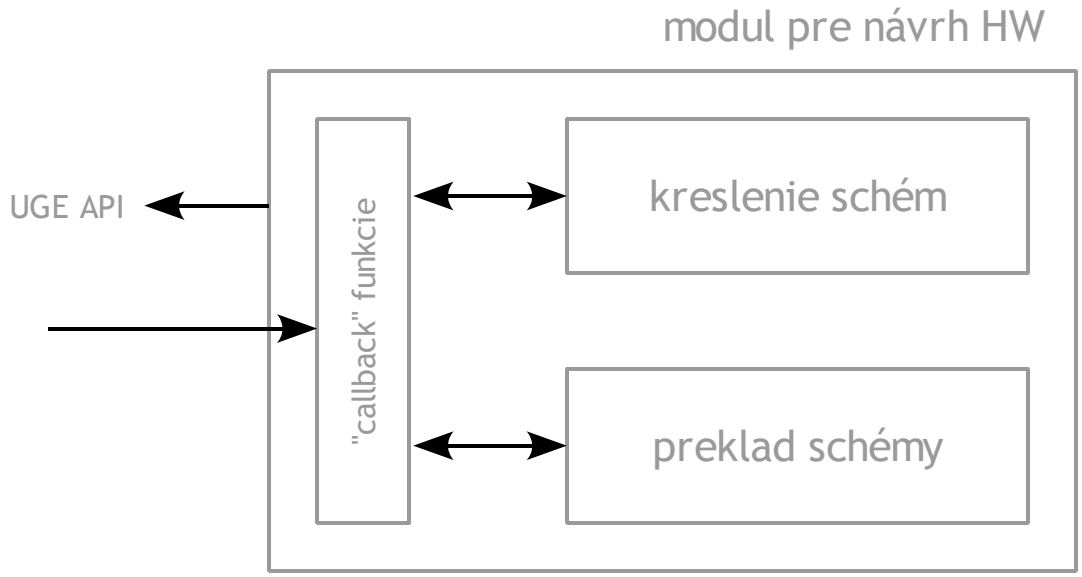
1. funckia pre nastavenie spätných volaní funkcií pluginu z jadra
2. funkcie spojené s uživateľským rozhraním
    - úprava hlavného menu (pridávanie, mazanie a nastavovanie položiek pre plugin)
    - nastavovanie položiek roletového menu pre jednotlivé grafové objekty
    - zobrazovanie oznamovacích a interaktívnych dialógov uživateľovi
3. funkcie pre prácu s vlastnosťami grafových objektov:
    - zistenie typu objektu podľa zadaného identifikátoru
    - získanie a nastavenie mena objektu
    - získanie identifikátoru objektu podľa zadaného mena
    - získanie pozície objektu podľa zadaného identifikátoru
    - zistenie typu uzlu, typu hrany a atribútov portu
    - pridávanie nových vlastností základných typov (celé číslo, číslo v pohyblivej rádovej čiarke, reťazec, logická hodnota) objektom
    - nastavenie a zísaknie hodnôt pridaných vlastností grafových objektov
4. funkcie pre prácu s logickou štruktúrou grafu:
    - vytváranie a rušenie grafových objektov (uzly, hrany, porty)
    - zmena typu a mena grafových objektov
    - nastavenie zdrojového a cieľového portu hrany
5. funkcie pre prácu s grafickou podobou objektov:
    - vytvorenie/mazanie/posun grafickej reprezentácie objektov
    - zmena grafických vlastností objektov (farba, výplň, veľkosť a typ písma, ...)
6. funkcie pre získavanie informácií o štruktúre grafu
    - počet všetkých uzlov grafu
    - identifikátory všetkých uzlov grafu
    - identifikátory všetkých hrán pripojených k uzlu
    - identifikátory všetkých portov pripojených k uzlu
    - identifikátor zdrojového a cieľového uzlu, ku ktorému je hrana pripojená
    - identifikátory hrán pripojených k danému portu
    - identifikátor uzlu, ktorému daný port patrí

3 Modul pre návrh HW architektúry

Táto kapitola sa zaoberá procesom návrhu zásuvného modulu pre program Univerzálny grafický editor, ktorý bude umožnovať kresliť schémy HW návrhu a poskytovať ďalšie užitočné funkcie v podobe generovania zdrojového kódu návrhu. Zásuvný modul bude využívať prostredie programu UGE popísaného v kapitole 2, ktorý poskytuje prostriedky pre kreslenie a prácu s obecným grafom (viz. definíca 2.1). Ak budeme pri kreslení HW schém používať uzly obecného grafu pre zobrazenie komponent a hrany grafu pre zobrazenie vodičov, máme k dispozícií základné prostriedky, ktoré na tento účel potrebujeme.

3.1 Štruktúra modulu

V súvislosti s koncepciou UGE a jeho rozhrania pre komunikáciu so zásuvnými modulmi bol modul rozdelený na 3 hlavné časti – jeho štruktúra je vidieť na obrázku 3.1.



Obrázok 3.1: Bloková schéma modulu.

Jedna časť definuje chovanie grafu – vytváranie grafových objektov, pravidlá pre spájanie jednotlivých objektov, reakcie na uživateľské akcie apod. Druhá časť obsahuje prostredie pre preklad grafu do zdrojového kódu jazyka VHDL. Tretia časť obaluje vnútornú logiku modulu do funkcií, ktoré sú pri inicializácií napojené na signály generované jadrom (pre popis princípu

signálov viz. kapitola 2.4). Tieto funkcie slúžia pre spätné volania modulu z jadra (preto sú to tzv. „callback funkcie"), čím je modul informovaný o akciách prevádzaných s grafom a obecne o akciách prebiehajúcich v jadre. Za rozhraním, ktoré tvoria callback funkcie, je už modul postavený na objektovom návrhu.

3.2 Vytváranie schém

Pri kreslení HW schém sa používajú tri základné prvky – komponenty, vodiče, porty komponent. Pre potreby ukladania komplexných schém ako základných stavebných blokov (HW komponent) a naopak – pre možnosť interaktívne sa vnoriť do bloku alebo komponenty a podrobnejšie ju definovať – zavádzame ešte jeden prvok, ktorý nie je typický pre „papierový" návrh HW. Tento prvok označuje prípojné body (tj. vstupy alebo výstupy) bloku resp. schémy a budeme ho označovať pojmom terminál.

3.2.1 Reprezentácia a vlastnosti prvkov schémy

V kapitole 2.3 sme popísali prostriedky, ktoré program UGE poskytuje pre vytváranie grafov. S týmito prostriedkami si musíme vystačiť aj pri kreslení HW návrhu. Pre reprezentáciu prvkov schém ich budeme využívať nasledovne:

- HW komponenty a bloky – uzly obecného grafu
- vstupy a výstupy komponety (porty) – porty uzlu
- vodiče – hrany obecného grafu
- terminály – uzly obecného grafu

Každý z obecných grafových elementov má súbor vlastností, ktoré sú pevne definované (tzn. daná vlastnosť existuje vždy) – nazvime ich statické vlastnosti. Niektoré z týchto vlastností môže modul nastavovať, aby definoval chovanie daného elementu a niektoré môže využívať pre svoje vlastné účely (napr. meno elementu). Aby však modul mohol ukladať aj svoje údaje (o ktorých jadro nič nevie) a rozširovať tak funkčnosť, je tiež možné elementom pridávať vlastnosti nové (viz. kapitola 2.3) – nazvime ich dynamické vlasnosti. Nasleduje vymenovanie a popis jednotlivých vlasntostí prvkov HW schémy. Jedná sa len o vlastnosti, ktoré využíva modul, takže niesú zahrnuté všetky vlastnosti obecných elementov grafu. Okrem vymenovaných atribútov má každý prvok unikátny celočíselný identifikátor. Písmeno „D", ktoré majú niektoré atribúty za svojím menom, značí, že atribút je dynamický. O uloženie všetkých prvkov a ich vlastností, jak statických tak dynamických, sa stará jadro.

1. komponenta
   - meno – je unikátne v rámci grafu, používa sa ako názov inštancie určitej triedy komponenty
   - trieda – názov triedy komponenty (napr. and, mpx4, input_buffer, ...)
   - typ – určuje typ grafového uzlu (možnosti – jednoduchý, násobný, port), všetky komponenty sú typu „násobný", čo znamená, že hrany sa k nim pripájajú prostredníctvom portov (a nie je možné ich pripájať priamo)
   - uživateľský kód (D) – ukladá uživateľov popis komponenty (napr. popis chovania v jazyku VHDL)
   - úroveň (D) – definuje, či je komponenta elementárna alebo je to schéma uložená do databázy komponent (možnosti – jednoduchá, zložená)
   - pridávanie portov (D) – volitelná vlastnosť (nemusí existovať), povoluje alebo zakazuje pridávanie portov danej komponente. Aby bolo možné porty pridávať, musí existovať a explicitne to povolovať.

2. port
   - meno (D) – názov portu, je unikátny v rámci komponenty (statickú vlastnosť „meno" pre tento účel použiť nemôžme, pretože tá musí byť unikátna v rámci celého grafu)
   - skupina (D) – volitelná vlastnosť (nemusí existovať), ak je definovaná určuje názov skupiny, do ktorej port patrí – táto vlastnosť umožňuje zoskupovať porty do skupín, ktoré budú pri výstupe do VHDL kódu zobrazené ako vektory a nie ako samostatné porty
   - trieda – názov triedy portu (možnosti – port, konektor) – trieda „port" slúži pre pripojenie obyčajného vodiča, trieda „konektor" pre pripojenie zbernice s väčším počtom signálov
   - typ – určuje typ grafového uzlu (možnosti – jednoduchý, násobný, port), porty sú v jadre implementované ako uzly špeciálneho typu „port"
   - pozícia – určuje, na ktorej strane komponenty sa port nachádza (vpravo, hore, ...)
   - smer (D) – určuje, či je port vstupný, výstupný, vstupný aj výstupný alebo tzv. buffer

3. vodič
   - trieda – názov triedy vodiča (možnosti – vodič, zbernica), triedu „vodič" je možné pripojiť len k portom triedy „port" a triedu „zbernica" je možné pripojiť len k portom triedy „konektor"
   - typ – určuje typ hrany (možnosti – priama, lomená, ortogonálna, krivka), hrany použité v module sú momentálne výlučne ortogonálne

- počet signálov (D) – definuje, koľko signálov sa nachádza v zbernici (ak je vodič triedy „vodič" tak je hodnota atribútu 1)
- meno signálu[1..n] (D) – definuje meno signálu, existuje „n" týchto atribútov, kde „n" je počet signálov
- šírka signálu[1..n] (D) – definuje šírku signálu, existuje „n" týchto atribútov, kde „n" je počet signálov

4. terminál
   - trieda – názov triedy, všetky terminály sú triedy „terminál"
   - typ – určuje typ grafového uzlu (možnosti – single, multiple, port), všetky terminály sú typu „multiple", z čoho vyplíva, že majú port (práve jeden) ku ktorému sa pripájajú hrany

Terminály sú definované ako uzly s portom preto, aby boli všetky operácie s vodičmi a ich prípojnými bodmi konzistentnejšie. Keby sa totiž vodiče pripájali priamo k terminálu (tzn. k uzlu z pohľadu topológie) vyvstalo by tak niekoľko problémov, ktoré by bolo treba explicitne ošetrovať. Napríklad by bolo nutné pridať uzlom triedy „terminál" vlastnosti prvku port (smer, skupina, ...) a pod. Takýmto problémom sa však pri navrhnutom riešení vyhneme. Z výčtu atribútov vidíme, že prvok terminál nemá žiadne špecifické vlastnosti – nositeľom všetkých potrebných informácií je práve jeho port. V podstate je terminál len akýsi obal pre port, ktorý nemôže stáť v grafe ako samostatný objekt.

Atribút „uživateľský kód" u prvku komponenta umožňuje uživateľovi ukladať textové dáta (popisujúce napr. chovanie komponenty), ktoré môžu byť využité pri transformácií schémy HW návrhu na dáta slúžiace inému systému. My budeme tento atribút využívať pre uloženie popisu komponenty v jazyku VHDL pri preklade schémy do zdrojového kódu jazyka VHDL. O tejto problematike pojednáva kapitola 3.4.

Atribút „smer" prvku port určuje typ portu z hladiska HW návrhu. Svoje vstupné porty môže komponenta využívať len na prijímanie dát, nie je možné do nich zapisovať. Na výstupné porty môže komponenta dáta zapisovať, ale nemôže z nich čítať. Typ „buffer" označuje porty, ktoré slúžia na výstup dát z komponenty, ale zároveň môže komponenta tieto dáta aj čítať. Štvrtý typ portu umožnuje dáta zapisovať aj čítať a to jak vlastnej komponente, tak aj ostatným prvkom na port pripojeným. Implementácia tohto atribútu je vhodná pri kreslení HW schém, kde je tak možné graficky odlíšiť jednotlivé typy portov, ale je nevyhnutná z pohľadu prekladu schémy do kódu jazyka VHDL.

Aby mal uživateľ možnosť prepájať komponenty aj zložitejšími zväzkami (budeme im obecne hovoriť zbernice) ako obyčajnými vodičmi, prvok vodič obsahuje atribút „počet signálov". Tento atribút definuje koľko signálov sa na zbernici nachádza. Pre „n" signálov

potom existuje „n" dvojíc dynamických vlastností - „meno signálu" a „šírka signálu". „Meno signálu" definuje jeho názov a „šírka signálu" definuje počet bitov, z ktorých daný signál pozostáva. Týmto spôsobom je zabezpečená možnosť prepájania komponent komplexnými zbernicami s väčším počtom rôznych signálov o rôznej bitovej šírke. Táto vlastnosť sa taktiež využíva najmä pri preklade schémy do VHDL kódu.

Uživateľ nemá možnosť priamo meniť všetky vymenované vlastnosti – u všetkých prvkov si vlastnosti „trieda" a „typ" nastavuje sám modul a užívateľ prakticky ani nevie o ich existencií. Takisto vlastnosti „úroveň" u komponenty alebo „smer" a „pozícia" u portu sú nastavované automaticky modulom – to vyplýva už z princípu ich použitia.

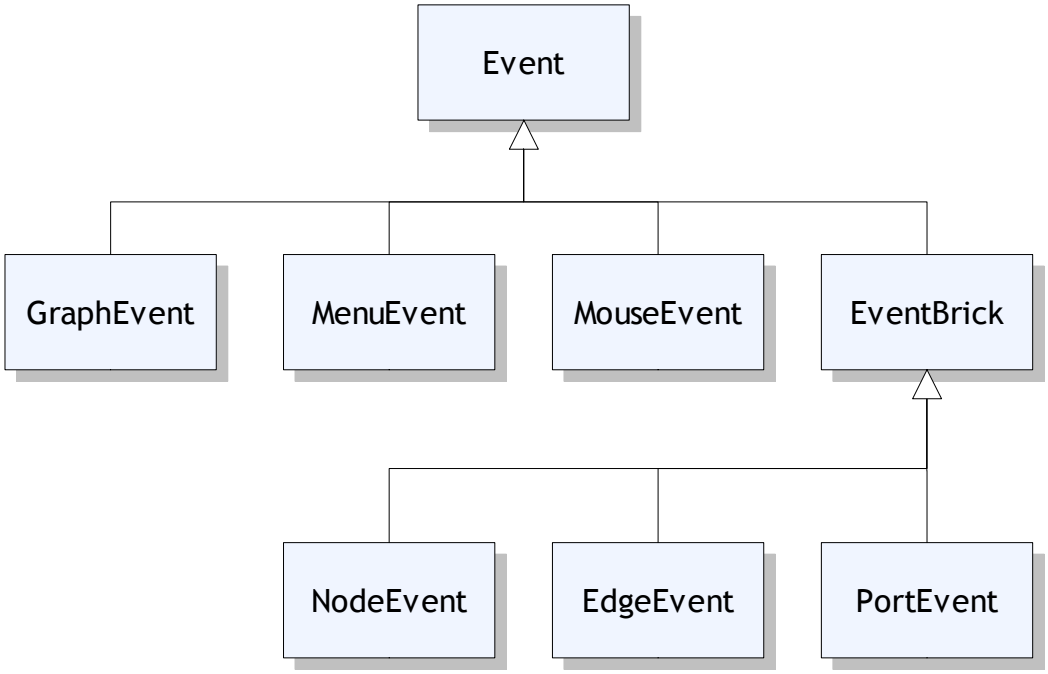
3.2.2 Obsluhovanie signálov jadra

Už vieme, aké prvky budeme pre kreslenie HW schém používať a ako budú tieto prvky reprezentované. Ale čo všetko môžeme s týmito prvkami robiť a ako sa to bude vykonávať? Program UGE sám o sebe nebude s našimi objektami robiť nič. Nevie ako sa má konkrétny objekt správať, ani ako má reagovať na akcie, ktoré uživateľ robí v uživateľskom rozhraní pri kreslení HW schém. Jadro UGE vykonáva nad grafom len implicitné operácie (ako napr. presúvanie grafových objektov apod.) a inak zachytáva akcie uživateľa a generuje príslušné signály. Je na nás, aby sme si na tieto signály napojili naše funkcie, ktoré budú akcie uživateľa obsluhovať (tj. vytvárať komponenty a ostatné grafové prvky, editovať ich vlastnosti, atď.). Predtým ako definujeme operácie, ktoré je možné pri kreslení HW schém prevádzať a to ako na ne bude modul reagovať si vymenujeme signály, ktoré modul zachytáva (tzn. signály, na ktoré má modul zaregistrované svoje funkcie).

- kliknutie na položku menu
- kliknutie na lavé tlačítko myši
- pustenie lavého tlačítka myši
- kliknutie na pravé tlačítko myši
- zmena vlastnosti grafového objektu
- nový graf
- zatvoriť graf

Pri vyvolaní signálu v jadre UGE a zavolaní príslušnej zaregistrovanej funkcie modulu sa predáva riadenie do modulu. Obsluhovanie signálov je založené na udalostiach – ak je zavolaná niektorá z funkcií modulu z jadra, znamená to, že v jadre nastala nejaká udalosť. Vytvorí sa objekt príslušnej triedy udalosti a tá na akciu uživateľa reaguje. Môže ísť o udalosť týkajúcu sa menu zásuvného modulu, uzlu, hrany alebo portu grafu, udalosť týkajúcu sa myši (kliknutie, prechod nad objektom, ...) alebo grafu obecne (uloženie, export, kliknutie na volnú plochu, ...).

Keďže jednotlivé udalosti majú viaceré spoločné vlastnosti, pri návrhu sme výhodne využili hierarchiu dedičnosti tried, viz. obr. 3.2.



Obrázok 3.2: Diagram dedičnosti tried udalostí

Trieda „Event", z ktorej sú zdedené všetky ostatné triedy udalostí ukladá identifikátor signálu, ktorý bol vyvolaný a identifikátor objektu, v súvislosti s ktorým udalosť vznikla. To zabezpečí, že bude možné jednoznačne identifikovať všetky predošlé udalosti. Nakoľko niektoré z nich spolu súvisia – napr. vybratie položky menu a následné kliknutie do grafu – je potrebné predchádzajúce udalosti ukladať, kým „nevyprší ich platnosť". K ich evidencii nám už teda stačí len ukladať inštancie tried do zoznamu predchádzajúcich udalostí.

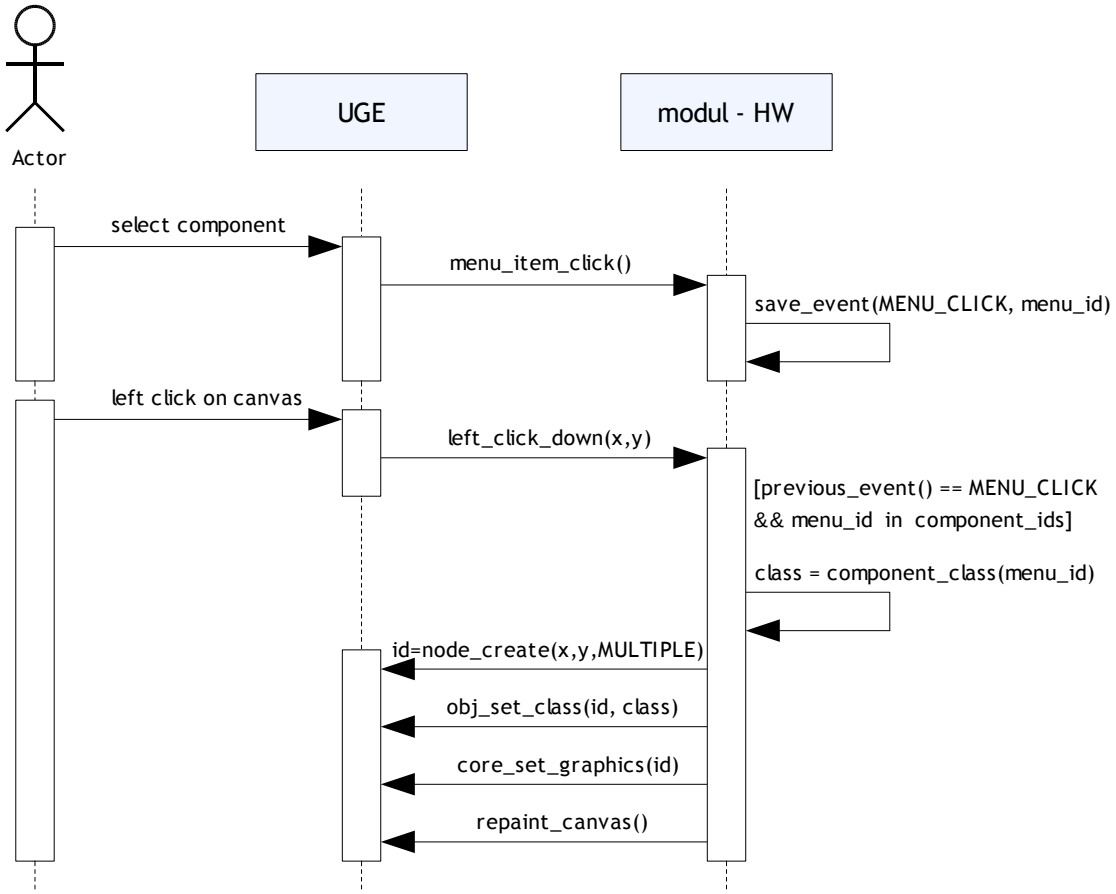
3.2.3 Operácie nad grafom

V tejto podkapitole si popíšeme, čo všetko je možné pri kreslení schém prostredníctvom modulu pre návrh HW robiť a ako tieto operácie modul prevádza (tzn. ako reaguje na udalosti vzniknuté v jadre UGE). Pre názornosť zložitejších operácií sú použité UML sekvenčné diagramy, na ktorých je najlepšie vidieť komunikáciu a postupnosť výmeny správ medzi jadrom a zásuvným modulom.

Predtým ako budeme popisovať samotné operácie, objasníme si, čo sa musí vykonať predtým ako je modul pripravený k použitiu. Pri aktivácií modulu z programu UGE jadro načíta

súbor s dátami modulu. Ten obsahuje grafickú reprezentáciu komponent s ich vlastnosťami a eventuálne ďalšie dáta. Modul vytvorí pomocou API funkcií UGE vlastné položky menu (ovládacie prvky pre uživateľa), medzi ktorými budú aj položky pre každú triedu (druh) komponenty z dátového súboru modulu. Nakoniec modul pripojí (zaregistruje) svoje funkcie na signály jadra uvedené v podkapitole 3.2.2. V tomto okamihu je všetko pre kreslenie schém pripravené.

Pridanie komponenty do schémy



Obrázok 3.3: Sekvenčný diagram pre operáciu pridanie komponenty

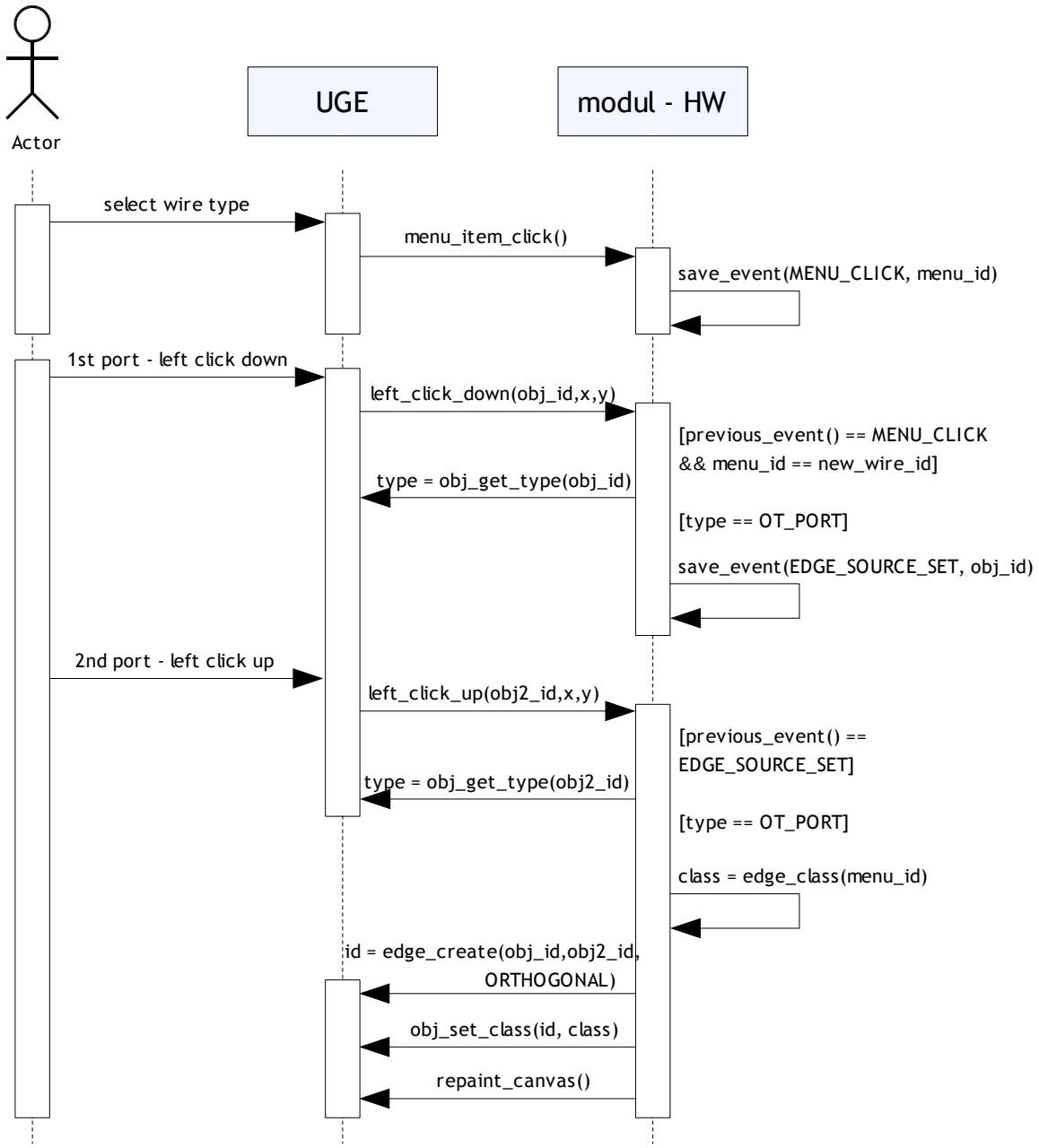
Pri pridávaní existujúcej komponenty z databázy komponent, uživateľ v menu zásuvného modulu zvolí príslušnú komponentu. Jadro zaznamená kliknutie do menu a zavolá registrovanú funkciu modulu. Táto funkcia vytvorí inštanciu triedy „MenuEvent" a tá udalosť obslúži – uloží sama seba do zoznamu vzniknutých udalostí. Inštancia triedy „MenuEvent" nesie identifikátor

signálu („MENU_CLICK”) a identifikátor položky, na ktorú bolo v menu kliknuté („menu_id”). Po tom ako uživateľ klikne na plochu grafu, je vyvolaný signál pre stlačenie lavého tlačítka myši a zavolaná príslušná funkcia modulu. Funkcia vytvorí inštanciu triedy „MouseEvent”, ktorá túto udalosť spracuje – zistí, či bola predchádzajúca uživateľova akcia kliknutie do menu a či položka, na ktorú bolo kliknuté, je niektorá z položiek pre pridanie existujúcej komponety. Ak sú tieto podmienky splnené, modul pomocou API funkcie programu UGE vytvorí uzol na pozícií, kam uživateľ klikol a nastavý tomuto uzlu príslušnú triedu (podľa položky, na ktorú uživateľ klikol). Potom dá opäť volaním API funkcie jadru UGE pokyn, aby nastavilo grafickú reprezentáciu a vlastnosti pridaného uzlu podľa hodnôt, ktoré načítalo z dátového súboru modulu pri inicializácii modulu. Nakoniec dá modul jadru pokyn prekresliť plochu grafu a tým je pridávanie komponenty dokončené.

Prepojenie portov komponent (vytvorenie vodiča)

Pri prepájaní komponent uživateľ taktiež vyberie z menu príslušnú položku pre vytvorenie vodiča alebo zbernice. To spôsobí volanie registrovanej funkcie modulu, ktorá túto udalosť uloží do zoznamu vyvolaných udalostí. Následne uživateľ klikne lavým tlačítkom na port komponenty, ku ktorému chce vodič pripojiť. Ak nastane akákoľvek iná udalosť (napr. kliknutie na pravé tlačítko myši) je akcia pridávania hrany zrušená (zoznam predošlých udalostí sa vyprázdni). Ak by bola pre kliknutie pravého tlačítka v danom okamihu definovaná nejaká akcia, bola by vykonaná. Po kliknutí na port je opäť zavolaná funkcia modulu napojená na stlačenie ľavého tlačítka a modul kontroluje predchádzajúcu udalosť. Ak to bolo kliknutie do menu na položku pre pridanie nového vodiča alebo zbernice, modul sa jadra spýta na aký typ grafového objektu uživateľ klikol. Ak to bol port uloží si do zoznamu udalostí „nastavenie počiatku hrany” s identifikátorom objektu, ku ktorému sa hrana pripája. Uživateľ po kliknutí na prvý port tlačítko nepustil, a premiestňuje kurzor myši na port cieľový. Nad ním tlačítko myši pustí – je zavolaná funkcia registrovaná na pustenie lavého tlačítka myši. Tá vytvorí ďalšiu udalosť (tj. inštanciu príslušnej triedy), ktorá zkontroluje či predchádzajúca udalosť bola „nastavenie počiatku hrany”. Ak áno skontroluje typ objektu, nad ktorým nastalo pustenie tlačítka myši. Ak to bol očakávaný typ objektu (tj. port) modul zavolá API funkciu pre vytvorenie ortogonálnej hrany medzi dvoma objektami (príslušnými portami) a nastaví jej príslušnú triedu („vodič” alebo „zbernica”). Vykonanie akcie je opäť dokončené príkazom na prekreslenie grafovej plochy. Celý proces vymieňania správ a hodnôt medzi modulom a jadrom UGE názorne zobrazuje sekvenčný diagram na obrázku 3.4.

Pri tejto akcií sa ešte prevádza kontrola, ktorá zabraňuje uživateľovi pripájať zbernice na obyčajné porty a naopak – pripájať vodiče na porty typu konektor (táto akcia nie je pre zachovanie prehladnosti v diagrame znázornená).

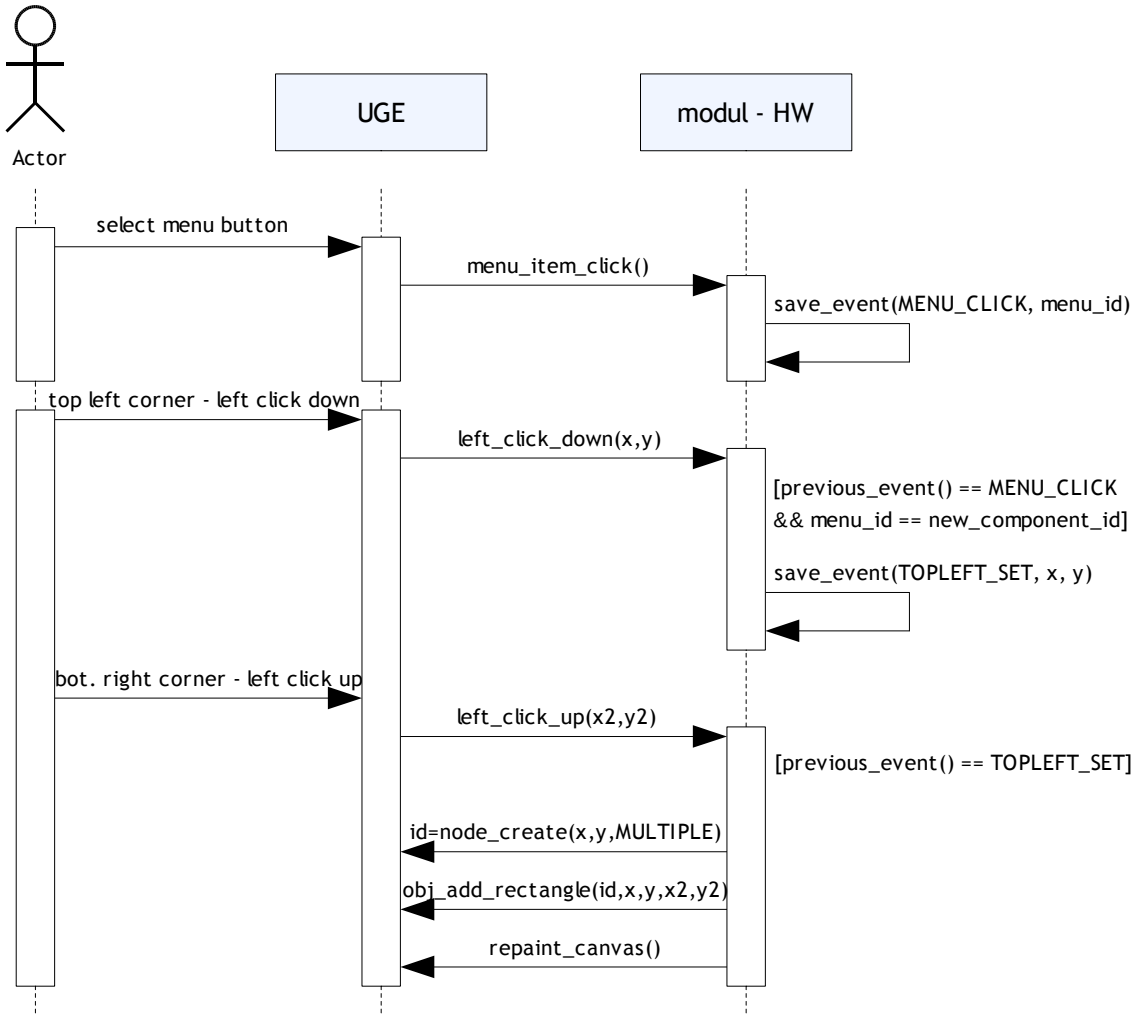


Obrázok 3.4: Sekvenčný diagram pre operáciu vytvorenie vodiča

Vytvorenie novej komponenty

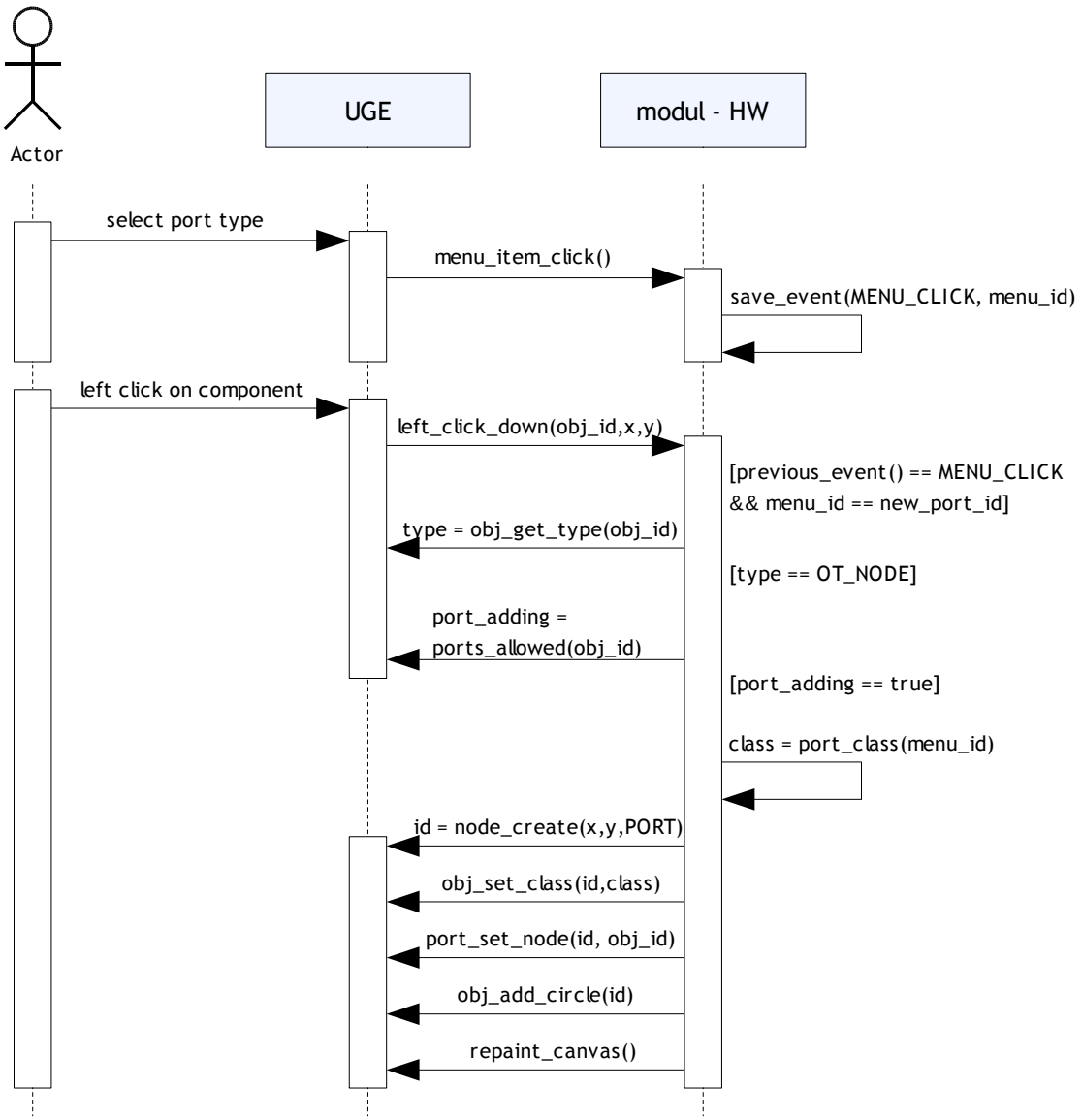
Uživateľ vyberie z menu položku pre vytváranie novej HW komponenty. Modul si uloží príslušnú udalosť do zoznamu vyvolaných udalostí. Následne uživateľ zmačkne ľavé tlačítko v oblasti grafu, ťahaním myši a pustením ľavého tlačítka zvolí oblasť novej komponenty. Modul

je o týchto akciách uživateľa opäť informovaný volaním príslušných zaregistrovaných funckií jadrom. Pri zmačknutí tlačítka modul kontroluje, aká bola predchádzajúca udalosť. Ak to bolo vybratie položky menu pre vytvorenie novej komponenty, uloží si do zoznamu udalosť „nastavenie pravého horného rohu komponenty", ktorá obsahuje súranice, na ktorých uživateľ tlačítko zmačkol. Pri pustení tlačítka modul opäť kontroluje typ predchádzajúcej udalosti – ak šlo o „nastavenie pravého horného rohu komponenty", modul dá jadru UGE pokyn na vytvorenie nového uzlu na daných súradniciach a pridá novému uzlu grafickú reprezentáciu – obdĺžnik. Akcia je dokončená prekreslením kresliacej plochy. Ďalej je na uživateľovi, aby novej kompoente nastavil požadované vlastnosti, prípadne jej pridal porty apod.



Obrázok 3.5: Sekvenčný diagram pre operáciu vytvorenie novej komponenty

Pridanie portu komponente



Obrázok 3.6: Sekvenčný diagram pre operáciu pridanie portu

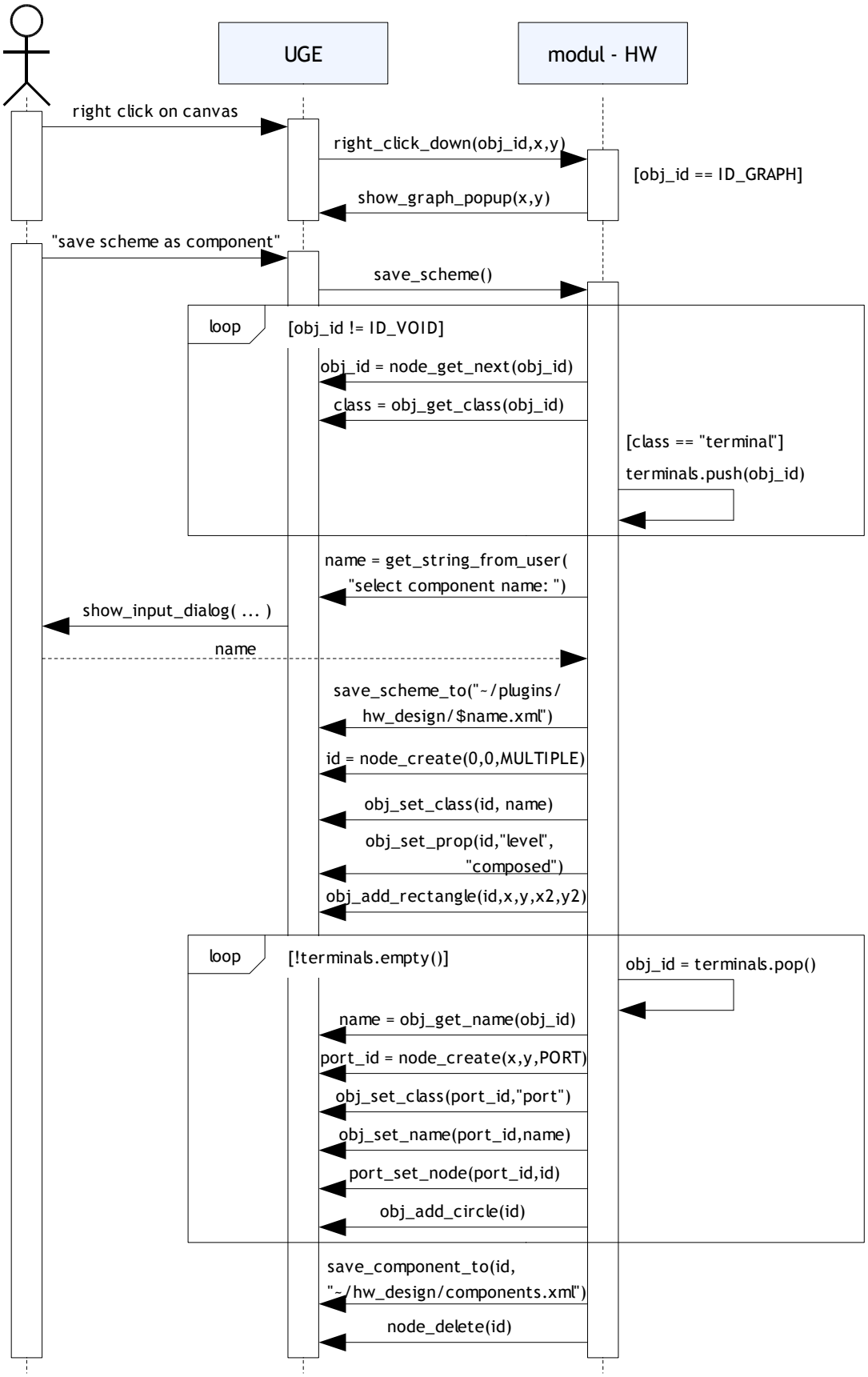
Rovnako ako pri predošlých operáciách, uživateľ zvolí z menu niektorú z položiek pre pridanie nového portu (existujú položky pre pridávanie štyroch rôznych typov portov a štyroch typov konektorov) a modul túto udalosť eviduje. Po následnom kliknutí na komponentu, ktorej chce uživateľ port pridať, modul kontroluje, aká udalosť tomu predchádzala. Ak to bolo kliknutie do menu na niektorú z položiek pridávania portov, modul skontroluje či objekt,

ktorému sa uživateľ snaží pridať port je naozaj grafový uzol. Keďže uzly grafu môžu reprezentovať aj také prvky, ktorým nie je možné porty pridávať (napr. terminály alebo základné logické komponenty – and, or, xor, atď.) musí pridávanie portov komponente explicitne povolovať vlastnosť „pridávanie portov". Ak je aj táto podmienka splnená, modul volaním API funkcie vytvorí port na pozícií pri okraji uzlu kam uživateľ klikol, nastavý mu príslušnú triedu (podľa položky menu, ktorú uživateľ zvolil) a pripadí mu vlastníka (komponentu). Obsluha akcie končí prekreslením plochy grafu, aby sa prevedené zmeny zobrazili uživateľovi.

Uloženie schémy ako komponenty

Aby sme návrhárovi HW umožnili kresliť schémy na rôznych úrovniach abstrakcie a zároveň týmito úrovňami prechádzať, bola navrhnutá možnosť ukladať celé schémy do databázy znovupoužitelných komponent. Takýmto spôsobom je potom možné abstrahovať komplexné schémy do obyčajných komponent a tieto ďalej používať ako stavebné prvky nových schém na vyššej úrovni. To umožnuje uživateľovi postupovať pri návrhu systému smerom zdola nahor. Aby mohol návrhár postupovať smerom zhora nadol, musí mať naopak možnosť interaktívne vstupovať do jednotlivých komponent a definovať ich „vnútro". Táto operácia bude popísaná až v nasledujúcej podkapitole.

Voľba na uloženie schémy ako komponenty sa uživateľovi zobrazí kliknutím pravým tlačítkom na prázdnu oblasť grafu. Modul na túto udalosť reaguje zobrazením kontextového menu, v ktorom sa požadovaná položka nachádza. Vstupnými a výstupnými bodmi (tj. portami) novej komponenty sa stanú terminály, ktoré sa pri ukladaní v schéme nachádzajú. Modul preto najskôr v cykle získa od jadra UGE všetky uzly grafu a u každého z nich zkontroluje vlastnosť „trieda". Ak je uzol triedy „terminál" uloží si modul jeho identifikátor do zoznamu. Po získaní všetkých terminálov, dá modul jadru pokyn na zobrazenie uživateľského dialógu pre zadanie mena novej komponenty. Po obdržaní mena volaním API funkcie uloží kompletnú schému do samostatného súboru v adresáry zásuvného modulu. Po tom, čo je schéma uložená, modul v grafe vytvorí nový uzol a nastavý mu triedu podľa mena, ktoré zadal uživateľ. Aby bolo u tejto novej komponenty možné identifikovať, že je to komponenta abstrahujúca schému, je jej nastavená vlastnosť „úroveň" na hodnotu „zložená". Ďalej modul pridá uzlu grafickú reprezentáciu (veľkosť je volená podľa počtu terminálov). Ako posledný krok modul získa atribúty portov terminálov schémy (meno, triedu, smer, ...), vytvorí odpovedajúce porty a pridá ich novému uzlu. Modul vytvoril kompletnú komponentu, ktorá bude predstavovať danú schému a túto komponentu voláním API funkcie uloží do databázy komponent modulu. Následne túto komponentu z grafu zmaže. Keďže medzitým nebola prekreslená plocha, uživateľ vytvorenie a uloženie komponenty nepozoruje, napriek tomu, že fyzicky komponenta existovala.



Obrázok 3.6: Sekvenčný diagram pre operáciu uloženie schémy ako komponenty

Ak teraz uživateľ pridá do novej schémy komponentu, ktorú vytvoril týmto spôsobom, môže prostým zvolením položky kontextového menu zobraziť schému, ktorú komponenta abstrahuje.

Sekvenčný diagram k tejto operácií (obr. 3.6) je samozrejme do určitej miery zjednodušený. Ako už bolo naznačené aj v texte vyššie, sekvenčné diagramy k jednotlivým operáciám sú abstrahované na potrebnú úroveň a neobsahujú niektoré minoritné operácie a kontroly ako napr. rozlíšnie grafického vykreslovania portov podľa ich typu (vstupný, výstupný, ...), kontrolu jednoznačnosti niektorých mien a obdobné operácie, ktoré neboli pre názornosť popisu podstatné.

Zanorenie do komponenty

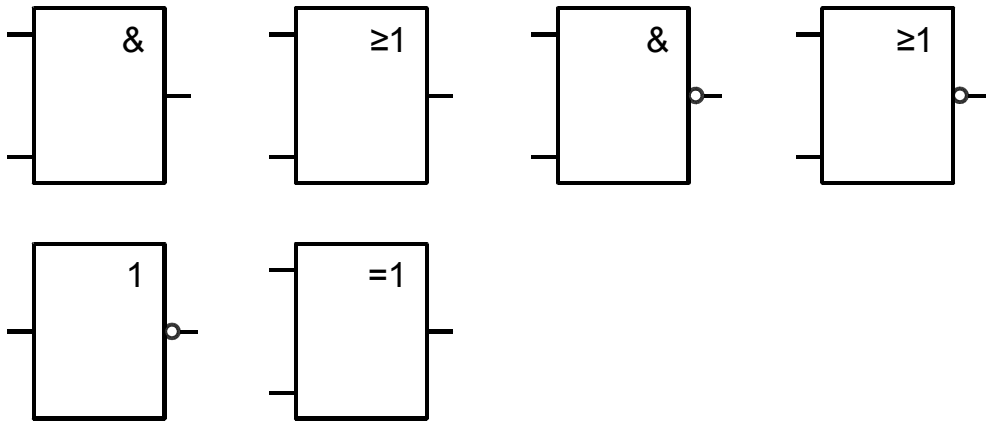
Ak si uživateľ vytvoril v schéme novú komponentu, môže ju na nižšej úrovni definovať opäť schémou. Ak má komponenta definovanú a nastavenú vlastnosť „pridávanie portov" (čo majú implicitne všetky uživateľom vytvorené komponenty), ktorá dovoluje komponente pridávať ďalšie porty, je tiež možné sa do tejto komponenty „zanoriť". Tzn. je možné definovať jej štrukúru alebo chovanie schémou. Túto akciu je možné vyvolať z kontextového menu komponenty. Po jej zvolení modul získa od jadra všetky porty komponenty a ich atribúty a otvorí nové okno pre schému. Do schémy pridá terminály, ktoré odpovedajú portom komponenty a nastaví im všetky potrebné atribúty (viz. podkapitola 3.2.1 – prvok port). Toto je výchozí bod pre uživateľa pri definovaní „vnútra" danej komponenty. Návrhárovi nie je zabránené, aby aj po tejto akcii pridával nové porty komponente alebo nové terminály do schémy, ktorá komponentu definuje. Tieto akcie sú medzi komponentou a schémou synchronyzované modulom.

Vybratie objektu a zrušenie výberu

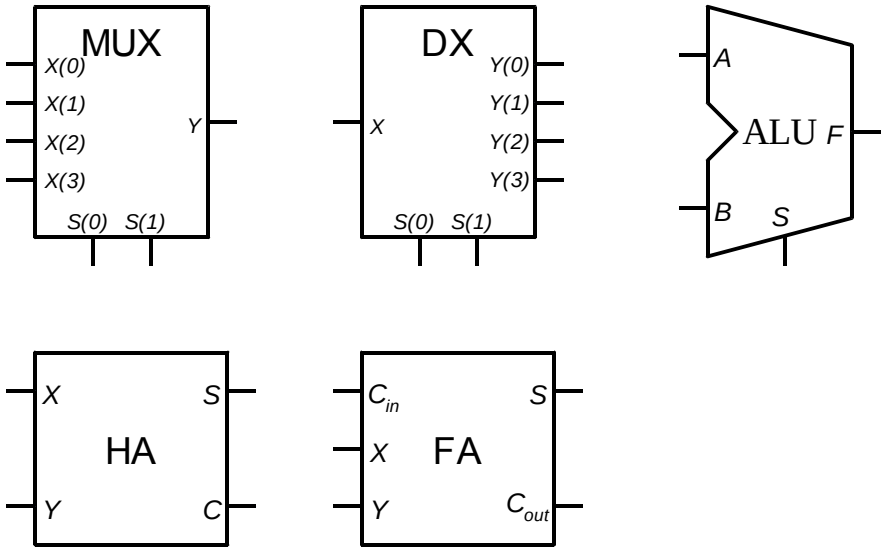
Jedna z triviálnejších operácií je vybratie objektu schémy. Ak uživateľ klikne lavým tlačítkom na objekt a modul pri obsluhe tejto udalosti zistí, že predtým nebola spravená žiadna iná súvisiaca operácia (vybatie položky menu apod.), zmení farbu grafickej reprezentácie objektu. Tým bude uživateľ informovaný, že objekt, na ktorý klikol vybral a je aktívny. Modul si uloží do zoznamu udalostí vybratie tohoto objektu a pri nasledovnej udalosti, v závislosti od jej druhu, zistí či má výber objektu zrušiť. Ak bude touto udalosťou napr. kliknutie na iný objekt, výber aktuálneho objekt sa zruší a vyberie sa objekt nový. Ak to bude kliknutie na iný objekt so zmačknutou klávesou „Control", výber prvého objektu sa nezruší a pridá sa výber nového (signály pre obsluhovanie udalostí klávesnice zatial niesú v jadre UGE implementované, ale do budúcna sa s nimi počíta).

3.3 HW komponenty

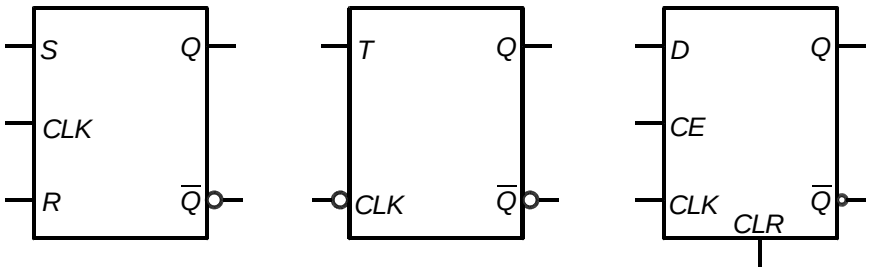
Pre reprezentáciu základných logických členov sme zvolili značky podľa štandardu IEEE. Okrem toho, že sú štandardizované, je výhodné aj to, že majú jednoduchú grafickú reprezentáciu.



Obrázok 3.7: Grafická reprezentácia vybraných základných logických členov, poľa poradia – and, or, nand, nor, not, xor

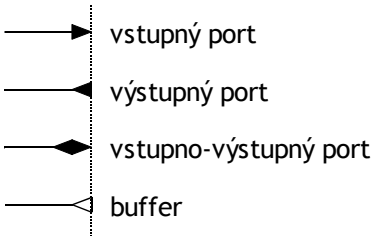


Obrázok 3.8: Grafická reprezentácia vybraných kombinačných obvodov – multiplexor, demultiplexor, aritmeticko-logická jednotka, polosčítačka, úplná sčítačka



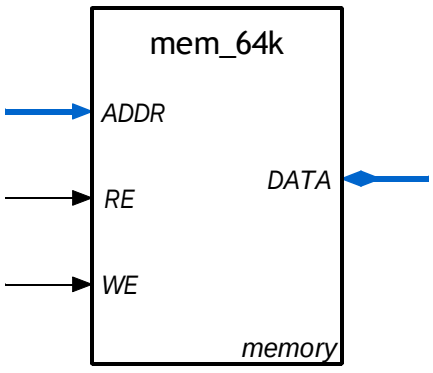
Obrázok 3.9: Grafická reprezentácia vybraných sekvenčných obvodov – klopný obvod RS, T a D s povolovacím a resetovacím vstupom

Pre optické rozlíšenie rôznych typov portov komponent, bola navrhnutá rozličná grafická reprezentácia pre každý typ portu, ako je vidieť na obrázku 3.10.



Obrázok 3.10: Grafická reprezentácia portov komponent

Na obrázku 3.11 je príklad uživateľom definovanej komponenty. Vlastnosť „meno" sa zobrazuje v pravom dolnom rohu komponenty a trieda komponenty pri jej hornom okraji. Zbernice a konektory majú narozdiel od obyčajných vodičov a portov modrú farbu pre ľahšiu vizuálnu identifikáciu.



Obrázok 3.11: Uživateľom definovaná komponenta

3.4 Preklad schémy do zdrojového kódu

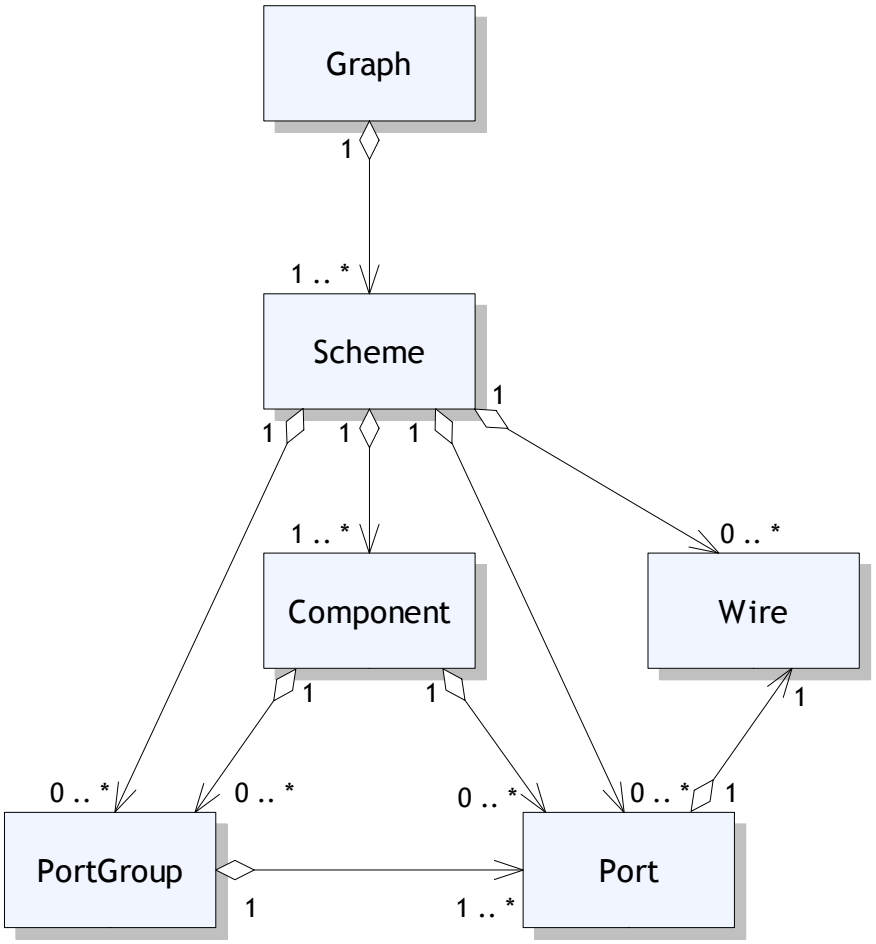
Ako bolo naznačené v úvode práce, z hľadiska modularity má schéma HW návrhu veľmi blízko k štruktúre zápisu HW návrhu v HDL jazykoch. To nám umožnuje použiť schému a dáta, ktoré schéma obsahuje, k vygenerovaniu zdrojového kódu návrhu pre konkrétny jazyk. Schéma resp. graf ako taký nesie len informácie o štruktúre návrhu, nie je možné ním zaznamenať popis chovania, procesy a podobné dynamické vlastnosti systému. Práve preto sme uživateľovi umožnili ukladať k HW komponentám aj ich popis (viz. podkapitola 3.2.1). Táto možnosť je tu primárne práve pre ukladanie reálneho kódu popisujúceho chovanie komponent (v terminológii návrhu HW sa jedná o behaviorálny popis). Ako cieľový jazyk bol zvolený VHDL.

3.4.1 VHDL

Pri generovaní výstupov zo schémy HW návrhu je nevyhnutné, aby modul poznal jej štruktúru a aby si mohol tieto dáta pred začatím samotného prevodu spracovať. Ale keďže inštanciu grafu má k dispozícií iba jadro UGE, modul si musí vytvoriť jeho obraz. Zámerne nehovoríme o kópií, pretože prostá kópia grafu by v tomto prípade nebola účelná. Postupným získavaním, identifikovaním a spracovaním jednotlivých grafových objektov modul vytvorí obraz schémy prispôsobený svojim potrebám. Tieto spracované dáta potom transformuje na požadovaný výstup.

Pre uloženie obrazu schémy modul používa hierarchiu tried, ktorá je zobrazená na obrázku 3.12. Každá trieda predstavuje určitú entitu grafu a má atribúty pre uloženie hodnôt vlastností určitého prvku HW schém. Preklad obrazu schémy do kódu VHDL zabezpečujú členské funckie týchto tried. Inštancia každej z týchto tried je pomocou nich schopná vytvoriť zo svojho obsahu VHDL kód, ktorý odpovedá jej významu v HW schéme. Vzhľadom k stromovej hierarchii tried, pre vygenerovanie kompletného kódu grafu stačí, ak je zavolaná funkcia najvyššie stojacej triedy „Graph". Tá sa potom postará o to, aby boli zavolané členské funkcie pre výstup pre všetky schémy, ktoré obsahuje. Každá zo schém zavolá členské funkcie pre výstup pre všetky svoje komponenty a vodiče, ktoré obsahuje atď. Svoj korektný preklad a výstup zabezpečuje každá trieda samostatne.

Vytvorenie obrazu grafu nie je uplne triviálna záležitosť, ako by sa mohlo zdať na prvý pohľad. Keď vychádzame z toho, že grafová plocha môze obsahovať viac nezávislých HW schém, potrebujeme nejakým spôsobom tieto schémy rozlíšiť. Tj. potrebujeme zistiť, ktoré prvky grafu sú nejakým spôsobom prepojené (a patria tak do jednej schémy). Na tento účel bol navrhnutý algoritmus prechádzania grafom, ktorý celým grafom prejde a nájde jednotlivé schémy a im náležiace prvky.



Obrázok 3.12: Prostredie pre preklad schémy - UML diagram tried

Algoritmus prechádzania grafom

Algoritmus pracuje celý čas len s celočíselnými identifikátormi grafových objektov. Jeho princíp spočíva v tom, že si na začiatku uloží do zoznamu všetky uzly grafu (tj. ich identifikátory). Potom zo zoznamu vyberie prvý uzol a ten považuje za počiatočný bod prvej schémy. Z neho začína prehľadávanie schémy – postupne algoritmus po všetkých hranách pripojených k uzlu prechádza do ostatným k nemu pripojených uzlov. Túto akciu opakuje rekurzívne v každom uzle, do ktorého sa dostane. Pritom kontroluje, v ktorých uzloch už bol a po ktorých hranách už prešiel, aby žiadny grafový objekt nespracoval 2x (pretože v tom prípade by vznikol v obraze grafu daný objekt duplicitne). Všetky uzly grafu, do ktorých sa už dostal odstraňuje zo zoznamu, ktorý vytvoril v prvom kroku. Ak už algoritmus prešiel po všetkých nájdených hranách, skončilo prehľadávanie prvej schémy – tá je teraz kompletná. Algoritmus skontroluje zoznam uzlov grafu, ktorý na začiatku vytvoril. Ak je prázdny, celý graf bol

spracovaný. Ak sa v ňom nachádza nejaký uzol, znamená to, že existuje ďalšia schéma a algoritmus ju opäť prechádza od počiatočného bodu atď., až kým nie je zoznam uzlov grafu vyprázdnený.

Algoritmus potrebuje pre ukladanie údajov tri pomocné premenné:

- zoznam načítaných uzlov, ktoré ešte neboli spracované
- zoznam načítaných hrán, ktoré ešte neboli spracované
- zoznam všetkých hrán, ktoré sme z jadra načítali

Kroky algoritmu:

- od jadra získame všetky uzly grafu a uložíme ich do zoznamu nespracovaných uzlov
- kým nebudú spracované všetky uzly zo zoznamu, vykonávame naledujúce operácie
  - zo zoznamu nespracovaných uzlov vyberieme prvý uzol a od jadra získame všetky k nemu pripojené hrany
  - hrany, ktoré sme doposiaľ nenačítali (tj. nenachádzajú sa v zozname všetkých načítaných hrán), uložíme do zoznamu všetkých načítaných hrán aj do zoznamu nespracovaných hrán
  - uzol sme v tomto okamžiku spracovali, a preto ho vymažeme zo zoznamu nespracovaných uzlov
  - kým nebudú spracované všetky hrany zo zoznamu nespracovaných hrán opakujeme cyklus
    - vyberieme prvú hranu zo zoznamu nespracovaných hrán a od jadra získame obidva uzly, ktoré spája
    - porovnaním so zoznamom nespracovaných uzlov zistíme, ktorý z týchto dvoch uzlov doteraz nebol spracovaný (kedže sme už získali danú hranu, jeden z jej uzlov sme už určite spracovali) – ak boli oba spracované odstránime hranu zo zoznamu nespracovaných hrán a pokračujeme ďalšou iteráciou cyklu
    - ak niektorý z dvoch uzlov spracovaný nebol, získame od jadra všetky k nemu pripojené hrany – tie, ktoré niesú v zozname všetkých načítaných hrán uložíme do zoznamu všetkých načítaných hrán aj do zoznamu nespracovaných hrán
    - tým sme spracovali ďalší uzol – vymažeme ho zo zoznamu nespracovaných uzlov
    - spracovali sme aj ďalšiu hranu – odstránime ju zo zoznamu nespracovaných hrán a pokračujeme ďalšou iteráciou cyklu
  - po skončení cyklu sme ukončili prechádzanie prvej schémy – všetky hrany, ktoré obsahuje sú uložené v zozname všetkých načítaných hrán (pred pokračovaním ďalšou iteráciou cyklu tento zoznam vyprázdnime)

Pri prechádzaní grafom modul postupne vytvára inštancie príslušných tried (schémy, komponenty, vodiče, porty, ...) a vytvára tak obraz grafu. Každá inštancia niektorej z tried sa pri vytvorení inicializuje – získa od jadra všetky potrebné vlastnosti prvku schémy, ktorý predstavuje. Všetky dáta, potrebné pri prachádzaní grafu, sa od jadra získavajú prostredníctvom funkcií vstupno-výstupného rozhrania postupne v cykloch (napr. získanie všetkých uzlov grafu, získanie všetkých hrán pripojenýchch k uzlu, atď.).

Prechádzanie grafu popísaným algoritmom nám zaručí, že:

- budú spracované všetky prvky nachádzajúce sa v grafe
- žiadny z prvkov grafu nebude spracovaný duplicitne
- bude korektne spracovaný aj graf obsahujúci cykly
- prvky grafu budú rozdelené do samostatných celkov (v našom prípade schém) podľa topológie grafu

3.4.2 Vysokoúrovňový model

Pri formálnej verifikácií metódou „model checking" sa verifikovaný model vytvára z návrhu HW. Tento model zapísaný vo verifikačnom nástroji má taktiež modulárnu štruktúru, ktorá môže byť podobná diagramu HW návrhu. To vyplýva z podstaty formálnej verifikácie – tej sú totiž podrobované určité entity návrhu. Tieto entity bývajú najčastejšie pri návrhu HW jeho funkčné bloky (komponenty). Z týchto dôvodov je možné uvažovať tiež o výstupe schémy HW návrhu modelu pre verifikačný systém. Opäť však musíme počítať s faktom, že zo štruktúry schémy nezískame žiadne informácie o procesoch, ktoré sa budú s verifikačným modelom prevádzať. Tieto dáta však môže poskytnúť uživateľ prostredníctvom atribútu „uživateľský kód", ktorý má každý funkčný blok schémy. Tie budú potom použité pri vytváraní výstupu schémy – modelu pre verifikáciu.

Prostredie pre preklad grafu bolo pôvodne navrhnuté pre potreby generovania VHDL kódu, ale ukázalo sa, že je dostatočne obecné pre získanie ľubovolných výstupov zo schémy HW návrhu. Najdôležitejšiu časť tohto prostredia tvorí algoritmus prechádzania grafom a triedy, ktoré uchovávajú obraz grafu v module. Tieto dve časti prostredia sa neviažu špeciálne ku generovaniu jedného druhu výstupu. Transformáciu a výstup grafu zabezpečujú členské funkcie tried, ktoré uchovávajú obraz grafu. Ak sa implementujú členské funckie týchto tried, ktoré budú generovať syntax určitého jazyka resp. výstup pre nejaký systém, je možné spolu s dátami, ktoré do grafu vložil uživateľ, generovať požadované výstupy. Vzhľadom k objektovému prístupu, pomocou ktorého je prostredie navrhnuté a implementované, je možné zdediť triedy z tohto prostredia do nových tried a týmto implementovať členské funkcie. To zabezpečí väčšiu prehladnosť kódu. Týmto spôsobom je možné zo schémy HW návrhu získať ľubovolný výstup (pre ľubovolný systém), s blokovou resp. modulárnou štruktúrou.

4 Implementácia modulu

Keďže zásuvný modul pre návrh HW je postavený na objektovom návrhu, pre implementáciu bol vybraný jazyk C++, ktorý toto paradigma podporuje. Okrem samotného programovacieho jazyka je používané aplikačné rozhranie (API) systému UGE, ktoré poskytuje funkcie popísané v podkapitole 2.5. Vďaka tomu je možné zásuvný modul skompilovať na akejkoľvek platforme, kde je implementovaný prekladač jazyka C++ a skompilovaná knižnica programu UGE. Žiadne iné prostriedky niesú pre funkčnosť modulu potrebné. Cieľom prekladu modulu je dynamická knižnica, ktorú bude využívať Univerzálny grafický editor.

Štruktúta zdrojového kódu je rozdelená, podobne ako návrh, do troch častí – komunikácia s jadrom UGE, obsluhovanie udalostí, prostredie pre preklad schémy. Graficky znázornené usporiadanie je možné nájsť v prílohe práce.

Základnú časť modulu tvorí súbor „hwdesign.cc", ktorý obsahuje deklarácie funkcií pre identifikáciu a inicializáciu modulu. Tieto funkcie sú pre každý zásuvný modul povinné a bez nich nie je modul funkčný. Hlavičky týchto funkcií sú deklarované tak, aby boli preložené a linkované ako funkcie jazyka C (napriek tomu, že zdrojový kód je kompilovaný prekladačom C++). Toto momentálne vyžaduje jadro UGE, aby mohlo tieto funkcie zavolať. Vo funkcii pre inicializáciu modulu, modul registruje funkcie pre spätné volania na konkrétne signály jadra a vytvára v uživateľskom rozhraní svoje vlastné menu. Do časti, ktorá komunikuje s jadrom, ešte patria súbory obsahujúce deklaráciu a definíciu funkcií pre spätné volania. V telách týchto funkcií sa na základe jadrom generovaných signálov vytvárajú inštancie tried udalostí, ktoré potom riadia správanie programu.

Druhá časť modulu implementuje triedy udalostí. Deklarácie tried sú kvôli prehladnosti a jednoduchej orientácií v hlavičkovom súbore „events.h" a každá trieda je potom definovaná v samostatnom súbore.

Tretia časť implementuje triedy prekladu grafu. Deklarácie tried sú v hlavičkovom súbore „translation.h" a každá trieda je, podobne ako u tried udalostí, definovaná v samostatnom súbore.

4.1 Spracovanie udalostí

Vzhľadom k princípu spracovania udalostí modul potrebuje pri vyvolaní novej udalosti poznať, aký typ udalosti nastal predtým. Pre tento účel ukladá všetky vytvorené objekty udalostí do zoznamu, implementovaného ako spojový zoznam ukazateľov na triedu „Event". Keďže „Event" je predchodcom všetkých tried udalostí, typ ukazateľ na triedu „Event" môže ukazovať na ktoréhokoľvek jej potomka. Nasleduje popis všetkých tried udalostí.

Event

Je to bázová trieda, z ktorej sú zdedené všetky triedy udalostí. Implementuje dva chránené atribúty spoločné pre všetky odvodené triedy – identifikátor objektu v súvislosti s ktorým udalosť vznikla a identifikátor signálu jadra, ktorý bol pri udalosti generovaný. Zároveň obsahuje členské funkcie pre nastavovanie a získavanie týchto atribútov.

GraphEvent

Táto trieda implementuje členské funkcie pre obsluhu udalostí súvisiacich s grafom – kliknutie na plochu grafu, zatvorenie, otvorenie a uloženie súboru s grafom, obsluha požiadavku na preklad grafu, atď.

MouseEvent

Táto trieda slúži na zaznamenávanie a ukladanie udalostí myši – kliknutie a pustenie tlačítka. K zdedeným atribútom triedy „Event" pridáva ešte atribút na uloženie pozície, kde bola udalosť s myšou vykonaná.

MenuEvent

Slúží na obsluhovanie udalostí spojených s vyvolaním položiek menu.

EventBrick

Virtuálna bázová trieda, ktorá slúži ako základná trieda pre všetky udalosti grafových objektov – uzlov, hrán a portov. Táto trieda pridáva čisto virtuálne metódy pre obsluhu udalostí, ktoré sú spoločné pre všetky tri typy grafových objektov. Konkrétne ide o metódy – pridanie a zrušenie grafového objektu, zmena mena objektu, zobrazenie kontextového menu pre objekt. Obsahuje aj ďalšie metódy spoločné pre grafové objekty, tieto už však niesú čisto virtuálne, tzn. že trieda „EventBrick" obsahuje aj ich definíciu. Ide o vybratie a zrušenie výberu grafového objektu.

NodeEvent, PortEvent, EdgeEvent

Sú triedy zdedené z „EventBrick". Definujú všetky čisto virtuálne funkcie, ktoré „EventBrick" deklaruje a pridávajú ďalšie funkcie, ktoré sú už špecifické pre každý typ grafového objektu. Trieda „EdgeEvent" napríklad obsahuje členské funkcie pre obsluhu nastavenia zdrojového a cieľového portu hrany.

4.2 Preklad grafu

Pri obsluhe udalosti „generovanie kódu VHDL" modul vytvorí objekt triedy „Graph". Ten v konštruktore volá svoju členskú funkciu „parse_graph", ktorá implementuje algoritmus prechádzania grafom. Počas prechádzania grafu sa vytvárajú inštancie príslušných tried, ktoré reprezentujú jednotlivé prvky schémy. Keď je prechádzanie grafu algoritmom dokončené, v module existuje celý obraz nakresleného grafu. Trieda „Graph" zavolaním metódy „print_scheme", každej jednej schémy, ktorá bola v grafe nájdená, spustí reťazové volanie funkcií zabezpečujúcich výstup každého objektu (objekt v zmysle inštancie triedy), ktorý bol počas prechádzania grafu vytvorený. Nasleduje stručný popis implementácie jednotlivých tried.

Graph

Trieda, ktorá riadi celý proces prekladu. Existuje len jedna inštancia, ktorá obsahuje zoznam schém, ktoré sú v grafe nakreslené. Tento zoznam je implementovaný ako spojový zoznam ukazateľov na triedu „Scheme". Trieda má jednu metódu a tou je „parse_graph", ktorá implementuje algoritmus prechádzania grafom.

Scheme

Trieda „Scheme" obsahuje zoznamy všetkých prvkov danej schémy – komponenty, vodiče, terminály a skupiny terminálov. Každý zoznam je implementovaný ako spojový zoznam ukazateľov na príslušnú triedu – „Component" pre komponenty, „Wire" pre vodiče, „Port" pre terminály a „PortGroup" pre skupiny terminálov. Okrem mnohých metód pre úpravu topológie schémy, obsahuje trieda dve, ktoré stoja za zmienku – metódu „init" a „print_scheme". Metódu „print_scheme" voláme, ak chceme dať triede pokyn, že má vygenerovať svoj VHDL kód. Trieda musí byť metódou „init" inicializovaná ešte pred volaním metódy pre výstup kódu.

Component, Wire, Port

Tieto triedy ukladajú obraz odpovedajúcich prvkov schémy HW návrhu – komponenty, vodiča, portu. V atribútoch majú uložené ich vlastnosti, popísané v podkapitole 3.2.1. Každá z tried obsahuje špecifické metódy pre výstup VHDL kódu.

PortGroup

Trieda „PortGroup" je obraz skupiny portov HW schémy. Obsahuje spojový zoznam ukazateľov na porty, ktoré do danej skupiny patria. Porty sa do skupiny pridávajú na základe hodnoty vlastnosti „skupina" každého portu. Účelom triedy „PortGroup" je, aby sa porty, ktoré do nej patria, nezapisovali do kódu VHDL samostatne, ale ako „vektor" portov.

5 Záver

Predmetom tejto práce bolo navrhnúť a vytvoriť zásuvný modul, ktorý by komplexne pokrýval problematiku návrhu hardware – od kreslenia schém HW návrhu, cez možnosť vytvárania a ukladania HW komponent do knižníc, až po preklad vytvorených schém priamo do kódu jazyka pre popis hardware (VHDL). Zásuvný modul mal pre tento účel využívať prostriedky, ktoré pre kreslenie obecných grafov poskytuje program Univerzálny grafický editor.

Modul pre kreslenie schém plne využíva aj pokročilé možnosti, ktoré UGE ponúka. Navrhnutý modul neumožňuje uživateľovi len obyčajné kreslenie schém. Vývojár HW môže vytvárať schémy na rôznych úrovniach s rôznou úrovňou abstrakcie a interaktívne sa medzi týmito úrovňami pohybovať. Použitím troch základných grafových prvkov (uzol, hrana, port uzlu) bola vybudovaná celá sada prvkov (komponenty, vodiče, zbernice, porty komponent, vektory portov, konektory, terminály, ...) slúžiacich pre kreslenie HW schém. Keďže samotný program UGE je momentálne stále vo vývoji a zatiaľ chýba implementácia niektorých kľúčových funkcií (najmä podpora ukladania a načítania dát), ktoré návrh nášho modulu využíva, nebolo možné do odovzdania práce implementovať fungujúci modul. O to podrobnejšie sa práca zaoberá návrhom tej časti modulu, ktorá zabezpečuje kreslenie schém a technická správa môže slúžiť ako návod k naprogramovaniu tejto časti.

V práci sa podarilo navrhnúť obecné prostredie pre preklad schémy HW návrhu na dáta pre ľubovolný systém, založený, rovnako ako návrh HW, na modulánej štruktúre. Implementovaný bol preklad schémy do jazyka VHDL, ktorý by mal mať, spomedzi možných výstupov, najširšie využitie v praxi.

Pri modelovaní HW návrhu sa v prostredí UGE občas stretávame s chovaním, ktoré nie je typické pre kreslenie HW schém. Uživateľ by napr. mohol očakávať, že môže vodiče pripájať na iné vodiče. Takéto operácie však jadro UGE nedovoľuje a považuje ich za chybné. Tieto problémy sa však dajú vzhľadom k tomu, že modul môže do grafu zasahovať takmer neobmedzene, implementačne obchádzať. Ide však o pracné programovanie takýchto operácií priamo v module, keďže natívna podpora pre dané chovanie v jadre neexistuje.

Pri ďalšom rozvíjaní projektu môžme uvažovať o možnosti popisovať chovanie entít a procesy v HW návrhu pomocou stavových automatov. Realizáciu tejto idei by nám zjednodušil existujúci projekt [13], ktorý implementuje modul, obsahujúci obecné prostredie pre konečné riadenie a umožnuje jednoduché naprogramovanie podpory pre všetky druhy automatov založených na konečnom riadení. Priestor pre ďalšie rozširovanie je aj v oblasti prekladu schémy do VHDL kódu, nakoľko tento jazyk je pomerne komplexný, s množstvom možností a využití a modul momentálne implementuje len vlastnosti, ktoré sú najpoužívanejšie pri návrhu HW.

Podieľanie sa na projekte Univerzálny grafický editor nám prinieslo veľa skúseností, najmä z pohľadu vyvíjania väčšieho projektu, na ktorom pracuje skupina ľudí. Prínosom bolo tiež nadobudnutie nových znalostí, najmä princíp fungovania zásuvných modulov a dynamických knižníc a v neposlednom rade zoznámenie sa s podpornými nástrojmi ako je systém CVS pre správu verzií zdrojových súborov.

Detailnejší popis systému UGE ako obsahuje táto práca v druhej kapitole, je možné nájsť v bakalárskej práci [12], ktorá sa zaoberá jeho návrhom.

Literatúra

[1] Fučík, O.: Návrh číslicových systémů. 2003.

[2] Crha, L.: Seminář VHDL. 2004.

[3] MICHEL, P., SAUCIER, G.: Logic and Architecture Synthesis.
Amsterdam : ELSEVIER, 1991. xii, 337 s. ISBN 0-444-89023-8.

[4] BÉRARD, B., et al.: Systems and Software Verification : Model-Checking Techniques and Tools. Berlin : Springer, 2001. xii, 190 s. ISBN 3-540-41523-8.

[5] MATOUŠEK, P.: Symbolic Data Structures for Parametric Verification [PhD. Thesis].
Brno University of Technology, Faculty of Information Technology, 2005. vi, 99 s.

[6] BARNAT, J., et al.: Model checking in IPv6 Hardware Router Design.
Dokument dostupný na URL
http://www.cesnet.cz/doc/techzpravy/2002/ipv6hwdesign (január 2007)

[7] Kratochvíla, T., Řehák, V., Šimeček P.: Verification of COMBO6 VHDL Design.
Dokument dostupný na URL
http://www.cesnet.cz/doc/techzpravy/2003/translationver (január 2007)

[8] Holeček J., et al.: How to Formalize FPGA Hardware Design.
Dokument dostupný na URL
http://www.cesnet.cz/doc/techzpravy/2004/formal-fpga-design (január 2007)

[9] Overview of IEEE Standard 91-1984 : Explanation of Logic Symbols. Texas Instruments Incorporated, 1996. Dokument dostupný na URL (január 2007)
http://www.ee.ic.ac.uk/pcheung/teaching/ee1_digital/TI%20-%20IEEE%20Symbols.pdf

[10] ARLOW, Jim, NEUSTADT, Ila. UML a unifikovaný proces vývoje aplikací. Překlad Bogdan Kiszka. 1. vyd. Brno : Computer Press, 2003. xiii, 387 s. ISBN 80-7226-947-X.

[11] Graf (teorie grafů). Wikipedie - Otevřená encyklopedie. Dokument dostupný na URL http://cs.wikipedia.org/wiki/Graf_(teorie_grafů) (január 2007)

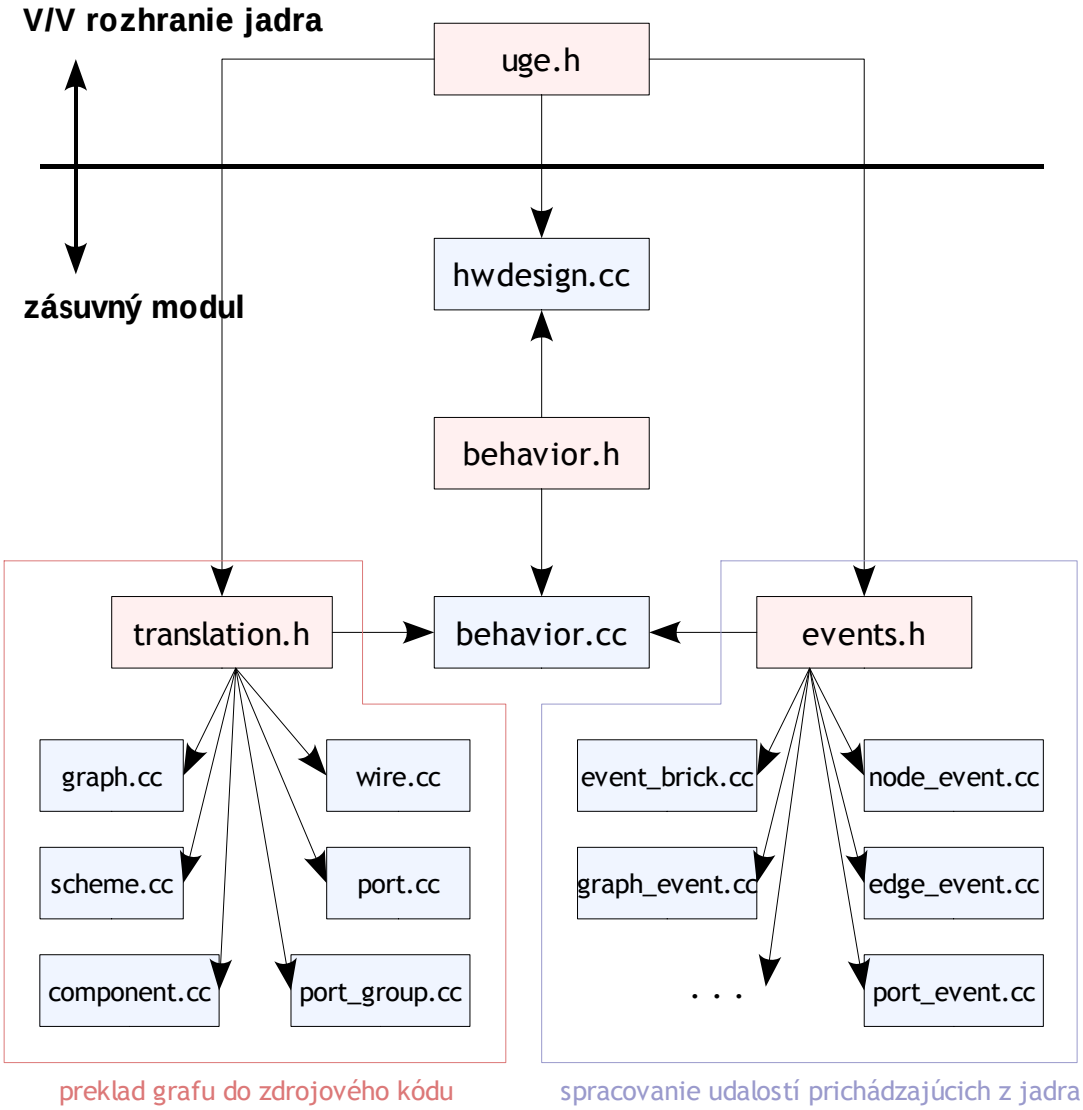
[12] Jadrný, M.: Univerzální grafický editor : V/V rozhraní [Bakalářská práce].
Brno : Vysoké učení technické, Fakulta informačních technologií, 2006. 39 s.

[13] Golich, P.: Univerzální grafický editor : Konečné automaty a Petriho sítě [Bakalářská práce]. Brno : Vysoké učení technické, Fakulta informačních technologií, 2006. 31 s.

Prílohy

Štruktúra zdrojových kódov

Červenou farbou sú označené hlavičkové súbory, ktoré obsahujú prototypy funkcií a deklarácie tried. Modrou farbou sú označené súbory s definíciami týchto tried. Smer šípiek označuje súbor, ktorý daný hlavičkový súbor vkladá. Umiestnenie zdrojových súborov modulu pre návrh HW v projekte UGE: ~/uge/src/plugins/hw_design



Obr. A: Štruktúra zdrojových kódov modulu pre návrh HW architektúry