

Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Jindřich Vodrážka

Vizualizace plánovacích domén

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Mgr. Lukáš Chrpa

Studijní program: Informatika, Správa počítačových systémů

2009

Na tomto místě bych chtěl poděkovat vedoucímu práce za poskytnutí tématu a odbornou pomoc při jeho zpracování. Dále bych chtěl poděkovat svým rodičům za podporu a trpělivost v průběhu celého studia, v jehož rámci byla práce napsána.

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

V Praze dne

Jindřich Vodrážka

Obsah

1	Úvod	6
1.1	Cíl práce	7
1.2	Motivace	7
1.3	Obsah práce	7
2	Úvod do problematiky	8
2.1	Definice pojmů	9
2.1.1	Plánovací doména	9
2.1.2	Použitelnost akce na stav	10
2.1.3	Přechodová funkce	10
2.1.4	Rozšířená přechodová funkce	10
2.1.5	Plánovací problém	11
2.1.6	Plán	11
3	Analýza existujících programů	12
3.1	itSIMPLE 2.0	12
3.2	GIPO III	13
4	Návrh řešení	14
4.1	Názvosloví	14
4.2	Prostředky vizualizace	15
4.3	Dekompozice problému	15
4.3.1	Vizualizace tříd a predikátových symbolů	16
4.3.2	Vizualizace plánovacích operátorů	17
4.3.3	Vizualizace plánovacího problému	18
5	Uživatelská dokumentace	20
5.1	Úvod	20
5.2	Popis rozhraní	20

5.2.1	Hlavní menu	22
5.2.2	Nástrojová lišta	23
5.2.3	Popis editačních ploch	24
5.3	Demonstrace	28
5.3.1	Svět kostek	28
5.3.2	Definice jazyka	29
5.3.3	Definice operátorů	32
5.3.4	Definice plánovacího problému	34
6	Programátorská dokumentace	37
6.1	Jazyk, knihovny a prostředí	37
6.2	Architektura programu	38
6.2.1	Convertor	38
6.2.2	EditWidget	39
6.2.3	DataWidget	42
6.2.4	DiagramWidget	42
7	Závěr	45
A	XML struktura	46
B	Obsah přiloženého CD	49
	Literatura	50

Název práce: Vizualizace plánovacích domén

Autor: Jindřich Vodrážka

Katedra (ústav): Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Mgr. Lukáš Chrupa

e-mail vedoucího: chrpa@ktiml.mff.cuni.cz

Abstrakt:

Předložená práce zkoumá možnosti vizualizace plánovacích domén. Náplní této práce je návrh grafické reprezentace pro plánovací domény a následné použití tohoto návrhu v programu, který bude fungovat jako grafický editor. Výsledná grafická reprezentace by měla být snadno převoditelná na klasickou reprezentaci plánovacích domén.

Program by měl umožňovat práci s plánovacími doménami v rozsahu typed STRIPS reprezentace. Měl by poskytovat uživateli možnost návrhu plánovací domény a její následný export do jazyka PDDL.

Klíčová slova: Plánovací doména, Plánování, Vizualizace, PDDL

Kapitola 1

Úvod

Následující úvod slouží pouze pro vytvoření přibližné představy o zkoumané problematice v širším kontextu. Přesnější definice použitých pojmů jsou uvedeny v další kapitole.

Plánování je jednou z klíčových oblastí ve výzkumu umělé inteligence. Schopnost naplánovat posloupnost akcí, jejichž provedení vede k nějakému předem specifikovanému cíli, je klíčová pro všechny inteligentní agenty. Plánovací doména představuje formální popis prostředí takového agenta. Tento formální popis je nutný pro realizaci vlastního plánování. Plánováním rozumíme v tomto kontextu proces, během kterého je vybrána určitá posloupnost akcí. Tyto akce jsou navíc v případě inteligentního agenta vybírány tak, aby vedly k nějakému cíli. Plánovací doména definuje zmíněné akce a umožňuje také přesně popsat podmínky, které představují pro agenta popis cíle.

Návrh konkrétní plánovací domény je náročný úkol. Pro každého agenta, který se specializuje na nějaký konkrétní problém je třeba navrhnout plánovací doménu tak, aby daný problém popisovala co nejlépe. Jeden problém může být popsán různými způsoby, které nemusí být stejně efektivní z hlediska následného použití v plánovacím procesu. Už v případě návrhu domény pro poměrně jednoduché prostředí je snadné udělat chybu, která může být příčinou logických rozporů při provádění plánu. Z výše uvedeného vyplývá, že je vhodné učinit proces návrhu co nejpřehlednějším.

1.1 Cíl práce

Práce si klade za cíl vytvoření grafického editoru, který umožní navrhovat a upravovat plánovací domény přehlednou formou. Tento editor bude podporovat vyjadřovací prostředky jazyka PDDL v rozsahu typed STRIPS. Umožní uživateli export do jazyka PDDL, takže s navrženými plánovacími doménami bude možné dále pracovat prostřednictvím externích plánovačů, jako je například **SGPlan** [3], nebo **SatPlan** [4].

1.2 Motivace

Pro návrh plánovacích domén lze v současné době využít programy jako jsou **GIPO** nebo **itSimple**¹. Tyto programy ale představují komplexní plánovací systémy, které umožňují mnohem víc než „jen“ navrhovat plánovací domény.

Specializovaný program zaměřený výhradně na návrh plánovacích domén není k dispozici. Pro popis domény v jazyce PDDL, můžeme využít libovolný textový editor a kontrolu přenechat na plánovači, který má textový popis problému v PDDL jako svůj vstup.

Textový soubor je jistě vhodným vstupem pro stroj, ale pokud nejde o literární dílo, rozhodně není přirozeným výstupem tvořivé lidské činnosti.

Při návrhu pomocí grafického editoru by bylo možné generovat popisy domén v jazyce PDDL mnohem rychleji a s menší náchylností k chybám. Tím je motivován vznik této práce.

1.3 Obsah práce

V kapitole pro uvedení do problematiky jsou uvedeny definice používaných pojmů včetně souvisejícího teoretického základu. Následně jsou rozebrány existující přístupy k problému v kapitole věnované analýze programů GIPO a itSIMPLE 2.0. Další kapitola popisuje návrh vlastního řešení a definuje syntaxi a sémantiku speciálně navržených diagramů pro vizualizaci plánovacích domén. Poté následuje uživatelská a programátorská dokumentace výsledného programu. V příloze je popis využití XML struktury a obsahu přiloženého CD, které je nedílnou součástí práce.

¹ Jejich rozboru je věnována třetí kapitola.

Kapitola 2

Úvod do problematiky

„Planning is the reasoning side of acting.”

M. Ghalab, D. Nau, P. Traverso

Plánování je proces, při kterém dochází k volbě a organizaci konkrétních akcí daného inteligentního agenta. Akce jsou vybírány tak, aby jejich provedením agent dosáhl předem stanoveného cíle. Tato práce se zabývá pouze klasickým plánováním [1]. To znamená, že prostředí agenta, které se snažíme popsat pomocí nějaké plánovací domény, má následující vlastnosti:

1. *plně pozorovatelné* - agent má k dispozici kompletní informaci o stavu prostředí
2. *deterministické* - agentem prováděné akce mají přesně definovaný výsledek
3. *konečné* - prostředí obsahuje konečný počet objektů
4. *statické* - změna prostředí je možná pouze jako důsledek akce provedené agentem
5. *diskrétní* - v objektech, akcích a jejich efektech

Prostředí splňující zmíněné vlastnosti lze chápat jako uzavřený systém, který se nachází v nějakém stavu. Pokud agent provede akci, změní stav systému. Akce jsou instantní - mají nulovou dobu trvání. Agent tedy dosáhne cíle, pokud se mu podaří nalézt posloupnost akcí, měnících stav systému do podoby vyhovující předem stanoveným podmínkám.

Plánovací doména představuje pro agenta popis prostředí ve kterém působí. Jsou v ní zachyceny informace o objektech prostředí a proveditelných akcích. Jak je vidět z následujících definic, je *plánovací doména* do značné míry abstraktní pojem. Tato abstrakce však umožňuje popsat nejružnější problémy¹. V rámci této práce je předložen způsob, jak lze tento abstraktní popis vyjádřit grafickou formou.

2.1 Definice pojmů

2.1.1 Plánovací doména

Mějme jazyk L predikátové logiky prvního řádu.

1. Definujme *atomy* a *plánovací operátory*:

- je-li $p(x_1, \dots, x_n)$ n -ární predikátový symbol a x_1, \dots, x_n termy jazyka L , pak $p(x_1, \dots, x_n)$ nazveme *atomem* jazyka L
- jsou-li c_1, \dots, c_n symboly pro konstanty jazyka L potom $p(c_1, \dots, c_n)$ nazveme *základním atomem* jazyka L
- n -ární *plánovací operátor* op nad jazykem L je definován uspořádanou čtveřicí:

$$(name(op), precondition(op), effect^+(op), effect^-(op))$$

kde:

- $name(op)$ je jméno operátoru ve tvaru: $op(x_1, \dots, x_n)$ s jednoznačným symbolem op a x_1, \dots, x_n - typovanými proměnnými jazyka L
- $precondition(op)$ je množina *atomů* jazyka L (*podmínky*)
- $effect^+(op)$ je množina *atomů* jazyka L (*pozitivní efekty*)
- $effect^-(op)$ je množina *atomů* jazyka L (*negativní efekty*)

2. Označme O množinu *plánovacích operátorů*.

Plánovací doménou nad jazykem L s operátory O nazveme uspořádanou trojici:

$$\Sigma = (S, A, \gamma)$$

kde:

¹V rámci klasického plánování jde převážně o problémy akademické - reálné prostředí často nesplňuje podmínky klasického plánování

- $S \subseteq 2\{t \mid t \text{ je základní atom jazyka } L\}$ je množina stavů
- $A = \{a \mid a \text{ je základní instance operátoru z } O\}$ je množina akcí
- $\gamma : (S \times A) \rightarrow S$ je přechodová funkce

2.1.2 Použitelnost akce na stav

Řekneme, že akce $a \in A$ je použitelná na stav $s \in S$ pokud:

$$precond(a) \subseteq s$$

Akci můžeme aplikovat, pokud v daném stavu s platí všechny podmínky které akce vyžaduje. Předpokládáme, že v množině $precond(a)$ nejsou žádné negativní atomy².

2.1.3 Přechodová funkce

$$\gamma : S \times A \rightarrow S$$

$$\gamma(s, a) = (s \setminus effect^-(a)) \cup effect^+(a)$$

kde $s \in S$ a $a \in A$ je akce použitelná na s .

2.1.4 Rozšířená přechodová funkce

Pro posloupnost akcí $\pi = \langle a_1, a_2, \dots, a_n \rangle$ délky n definujeme rozšířenou přechodovou funkci rekurzivně: $\gamma^* : S \times \pi \rightarrow S$

1. $n = 1 \Rightarrow \gamma^*(S, \langle a_1 \rangle) = \gamma(S, a_1)$
2. $n > 1 \Rightarrow \gamma^*(S, \langle a_1, \dots, a_{n-1}, a_n \rangle) = \gamma^*(\gamma^*(S, \langle a_1, \dots, a_{n-1} \rangle), a_n)$

²Negativní atomy v prekondici lze eliminovat zavedením nových predikátových symbolů jazyka (např. $\neg zapnuto$ nahradíme $vypnuto$)

2.1.5 Plánovací problém

Mějme plánovací doménu $\Sigma = (S, A, \gamma)$ (nad jazykem L).
Plánovacím problémem nazveme trojici:

$$P = (\Sigma, s_0, g)$$

kde:

- $s_0 \in S$ je počáteční stav systému
- g je množina *základních atomů* jazyka L . Každý stav $g' \in S$ pro který platí:

$$g \subseteq g'$$

nazveme cílovým stavem.

2.1.6 Plán

Máme-li zadán plánovací problém $P = (\Sigma, s_0, g)$, nazveme *plánem* takovou posloupnost akcí $\pi = \langle a_0, a_1, a_2, \dots, a_n \rangle \quad \forall i = 0, \dots, n : a_i \in A$, že:

$$g \subseteq \gamma^*(s_0, \pi)$$

kde γ^* je rozšířená přechodová funkce (2.1.4).

Kapitola 3

Analýza existujících programů

3.1 itSIMPLE 2.0

Program *itSIMPLE* [6] je komplexní nástroj pro návrh plánovacích domén. Kromě toho umožňuje jejich testování a řešení konkrétních plánovacích problémů pomocí externího plánovače. Implementace je provedena v jazyku Java, což zajišťuje výhodu přenositelnosti mezi nejrozličnějšími platformami. Pro interní reprezentaci dat využívá **itSIMPLE** jazyk XML a poskytuje uživateli možnost konverze do UML, PDDL a PNML. Pro návrh a vizualizaci plánovacích domén je využit jazyk UML.

Návrh plánovací domény v **itSIMPLE** je realizován z velké části pomocí *Classes diagramu*, kde má uživatel možnost nadefinovat typy objektů (pro každý typ je třeba vytvořit třídu - class). Asociace mezi třídami potom reprezentují binární predikátové symboly. Pro každou třídu lze nadefinovat operátory. Každý operátor představuje deklaraci nějaké akce. Dále lze v rámci třídy definovat atributy, které podle svého typu představují n-ární predikátový symbol nebo funkci. K definici akcí slouží *State machine diagram*. Jeden stavový diagram odpovídá vždy jedné třídě objektů. Uživatel zde má možnost nadefinovat stavy, do kterých se objekt dané třídy může dostat. K nadefinování akce potom slouží přechodové hrany mezi jednotlivými stavy, u kterých lze uvést podmínky a efekty. Máme-li nadefinované typy objektů a akce, je reprezentace množiny stavů už jenom otázkou vytvoření konkrétních instancí objektů, které jsou součástí navrhované domény. K tomuto účelu je v **itSIMPLE** použit diagram *Object repository*.

3.2 GIPO III

Prostředí *GIPO* [5] nabízí sadu nástrojů pro návrh a analýzu plánovacích domén. Implementace je provedena v jazyku Java. Vnitřní reprezentace dat je realizována pomocí OCL. Popis hotové domény lze převést také do PDDL. Podobně jako **itSIMPLE**, poskytuje i **GIPO** nástroje pro analýzu navržených plánovacích domén a umožňuje použití externího plánovače při řešení konkrétních plánovacích problémů.

Návrh plánovací domény v prostředí **GIPO** se obejde téměř bez vizualizace. Základní editační nástroje (*Sort view*, *Predicate view* a *State view*) umožní uživateli nadefinovat jednotlivé třídy (zde *Sorts*), predikátové symboly a seznam stavů pro každou třídu. Pomocí diagramu lze poté nadefinovat jednotlivé akce (Primitive transitions). Konkrétní objekty pro danou doménu se vkládají pomocí *Sort view*. Všechny provedené akce se ukládají ve formátu OCL. Alternativou k použití základních nástrojů je tzv. *Object Life History* editor. Zde má uživatel možnost ve speciálním diagramu zachytit stavové automaty pro každou třídu objektů. Diagram představuje grafickou reprezentaci orientovaného grafu se dvěma typy uzlů a třemi typy hran. Vzniklý diagram lze převést do OCL (třídy, predikáty, stavy jednotlivých tříd a akce jsou vygenerovány automaticky). Pro dokončení návrhu plánovací domény stačí už jenom vytvořit konkrétní instance nadefinovaných tříd pomocí *Sort view*.

Kapitola 4

Návrh řešení

4.1 Názvosloví

Vzhledem k tomu, že názvosloví použité ve výše zmíněných programech není jednotné, rozhodl jsem se pro přehlednost nadefinovat názvosloví vlastní. Význam klíčových pojmů používaných v následujícím textu bude následující:

- *Objekt* je konkrétní prvek, který je součástí popisované domény (např. agent)
- *Třída* je označení pro **množinu**, ve které sdružujeme všechny *objekty* společných vlastností
- *Proměnná* je zástupný symbol pro libovolný *objekt* v rámci dané *třídy*
- *Predikát* je *atom* jazyka L popisované domény (viz. definice 2.1.1)

Pro *třídy* platí **dědičnost**. Je-li R „*rodič*“ a $P_1 \dots P_n$ jeho „*potomci*“, potom platí následující tvrzení:

$$\forall i \in 1 \dots n : P_i \subseteq R \quad (4.1)$$

$$\forall i, j \in 1 \dots n, i \neq j : P_i \cap P_j = \emptyset \quad (4.2)$$

První z nich (4.1) říká, že potomci dědí vlastnosti po svém rodiči a druhé (4.2) zakazuje vícenásobnou dědičnost.

O *proměnných* říkáme že jsou *typované*. Pojmy *typ* a *třída* je možné zaměnit a v následujícím textu mají stejný význam: stanovují množinu objektů, jejíž prvky může proměnná zastupovat.

Argumenty *predikátů* mohou být *proměnné* nebo *objekty* různých *typů*, jejichž pořadí je ale pevně stanoveno. Dva predikátové symboly stejného jména a arity lze tedy rozlišit podle jakési „*signature*”.

Definice: Signatura predikátu

Nechť T je množina *tříd* a $p(x_1^{t_1}, \dots, x_n^{t_n})$ je *atom* jazyka L . Pokud $t_i \in T$ je *typ* argumentu x_i pro každé $i \in 1 \dots n$, potom uspořádanou n -tici (t_1, \dots, t_n) nazveme *signaturou predikátu* p .

4.2 Prostředky vizualizace

Pro grafické znázornění plánovací domény jsem se rozhodl po vzoru programu **itSimple** využít sadu diagramů. Nejde však o UML diagramy, jejichž vyjadřovací schopnosti jsou bezpochyby obrovské, ale o speciálně navrženou sadu diagramů určenou pro plánovací domény. Při návrhu byl kladen důraz na jednoduchost a zachování možností pro další rozšíření. Sémantika použitých diagramů je popsána v této kapitole.

4.3 Dekompozice problému

Při návrhu plánovací domény je nejprve třeba definovat nějaký jazyk L logiky prvního řádu (2.1.1). Jmenovitě potřebujeme stanovit *predikátové symboly* a *konstanty* jazyka L .

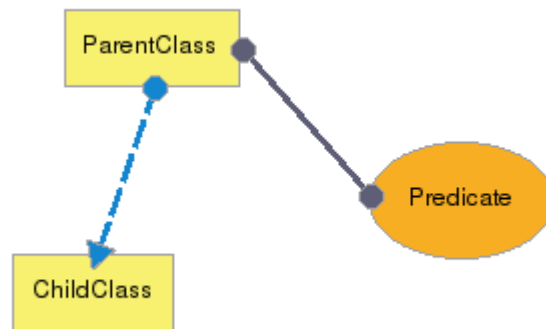
Dále je třeba zavést *třídy*¹ pro proměnné a objekty. V tomto kontextu odpovídají objekty konstantám jazyka L .

Aby byl návrh plánovací domény kompletní, je třeba dodat ještě popis *plánovacích operátorů*.

Výše uvedené požadavky postupně splníme v následujících třech krocích:

1. Deklarace *tříd* a definice *predikátových symbolů* pro jazyk plánovací domény.
2. Definice *plánovacích operátorů*.
3. Definice *plánovacího problému*.

¹definice plánovací domény zavedení tříd přímo nevyžaduje - tento požadavek vyplývá ze záměru práce, kterým je export plánovací domény do PDDL s podporou typů



Obrázek 4.1: Vizualizace tříd a predikátových symbolů

Po provedení kroků (1) a (2) získáme téměř kompletní definici plánovací domény. V jazyce L nám budou chybět pouze konstanty, které definujeme pomocí objektů v (3).

Samotná plánovací doména ovšem pro realizaci plánování nestačí. Proto v bodě (3) definujeme (s použitím objektů) plánovací problém, který je dle definice (2.1.5) tvořen uspořádanou trojicí (Σ, s_0, g) :

- plánovací doména Σ je na tomto místě již specifikována
- množiny s_0 a g můžeme vytvořit z výše definovaných predikátových symbolů tak, že jako argumenty dosadíme konstanty, právě definované v bodě (3)

Pro každý z výše uvedených kroků bude v následujícím textu popsán zvláštní diagram. Základem všech tří diagramů je neorientovaný graf se *dvěma druhy vrcholů* a *jedním typem hran*. Tento jednoduchý koncept je přehledný a snadno rozšiřitelný ².

4.3.1 Vizualizace tříd a predikátových symbolů

První typ diagramu se vztahuje ke kroku (1). Ilustrační příklad je na obrázku 4.1.

²Můžeme např. zavést orientované hrany, vhodným způsobem uzly graficky odlišit barvou, nebo obtažením atd.

Uzly: *třídy, predikáty*

Význam diagramu:

- uzel představující *třidu* je zobrazen jako **obdélník** obsahující název třídy
- uzel představující *predikát* je zobrazen jako **elipsa** obsahující název predikátu
- **neorientovaná hrana** mezi *třídou* a *predikátem* definuje typ argumentu tohoto predikátu
- **orientovaná hrana** mezi dvěma *třídami* upravuje dědičné vztahy (výchozí uzel je rodičem, cílový uzel potomkem)

Omezení:

- názvy definovných tříd jsou unikátní
- orientované hrany spolu s uzly, které propojují tvoří les (graf bez cyklů)
- všechny *predikáty*, které mají shodný název, mají také stejnou aritu

4.3.2 Vizualizace plánovacích operátorů

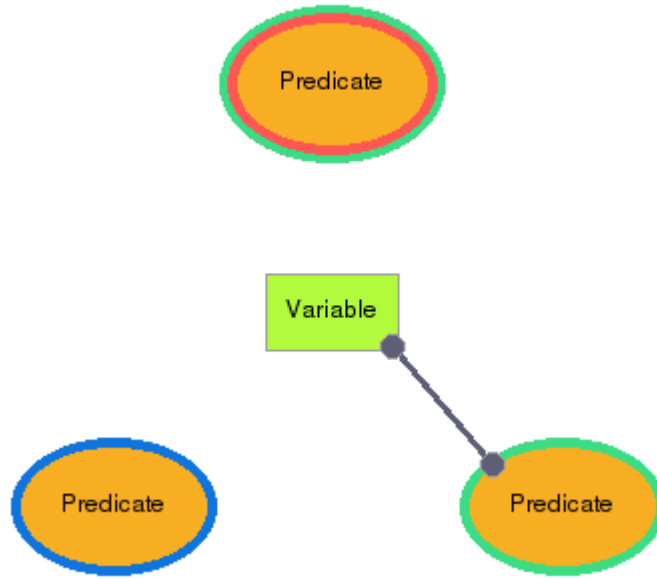
Druhý typ diagramu se vztahuje ke kroku (2). Ilustrační příklad je na obrázku 4.2.

Diagram umožňuje definici plánovacího operátoru. Pro účely následujícího popisu uvažujeme operátor Op .

Uzly: *proměnné, predikáty*

Význam diagramu:

- uzel představující *proměnnou* je zobrazen jako **obdélník** obsahující název proměnné
- uzel představující *predikát* je zobrazen jako **elipsa** obsahující název predikátu
- *predikát* p je obtažen **zeleně**, pokud $p \in precondition(Op)$
- *predikát* p je obtažen **červeně**, pokud $p \in effect^-(Op)$



Obrázek 4.2: Vizualizace plánovacího operátoru

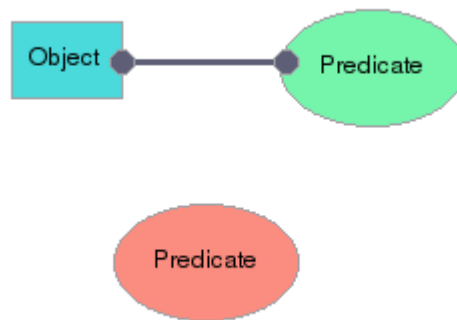
- *predikát* p je obtažen **modře**, pokud $p \in effect^+(Op)$
- **neorientovaná hrana** mezi *predikátem* a *proměnnou* symbolizuje dosazení proměnné do predikátu

Omezení:

- názvy proměnných jsou unikátní v rámci operátoru
- pro *predikát* p platí následující:
 - $p \in precondition(Op) \Rightarrow p \notin effect^+(Op)$
 - $p \in effect^+(Op) \Rightarrow p \notin effect^-(Op) \wedge p \notin precondition(Op)$
 - $p \in effect^-(Op) \Rightarrow p \notin effect^+$

4.3.3 Vizualizace plánovacího problému

Poslední diagram se vztahuje ke kroku (3). Ilustrační příklad je na obrázku 4.3.



Obrázek 4.3: Vizualizace plánovacího problému

Uzly: *objekty, predikáty*

Význam diagramu:

- uzel představující *objekt* je zobrazen jako **obdélník** obsahující název objektu
- uzel představující *predikát* je zobrazen jako **elipsa** obsahující název predikátu
- *predikát* p je vybarven **červeně**, pokud $p \in s_0$ (viz. definice 2.1.5)
- *predikát* p je vybarven **zeleně**, pokud $p \in g$
- **neorientovaná hrana** mezi *predikátem* a *objektem* symbolizuje dosažení objektu do predikátu

Omezení:

- názvy objektů jsou unikátní v rámci plánovacího problému

Kapitola 5

Uživatelská dokumentace

5.1 Úvod

Program VIZ je grafický editor určený pro návrh klasických plánovacích domén. Kromě návrhu plánovací domény umožňuje také definici plánovacích problémů. Navrženou doménu je možné exportovat do jazyka PDDL. Návrh domény je realizován skrz editaci diagramů se speciální sémantikou, která umožňuje zachycení informací potřebných k popisu domény.

Domény, které lze pomocí programu navrhnout, jsou omezeny na podmnožinu PDDL 1.2 [7]. Tato podmnožina nevyužívá následující klíčová slova:

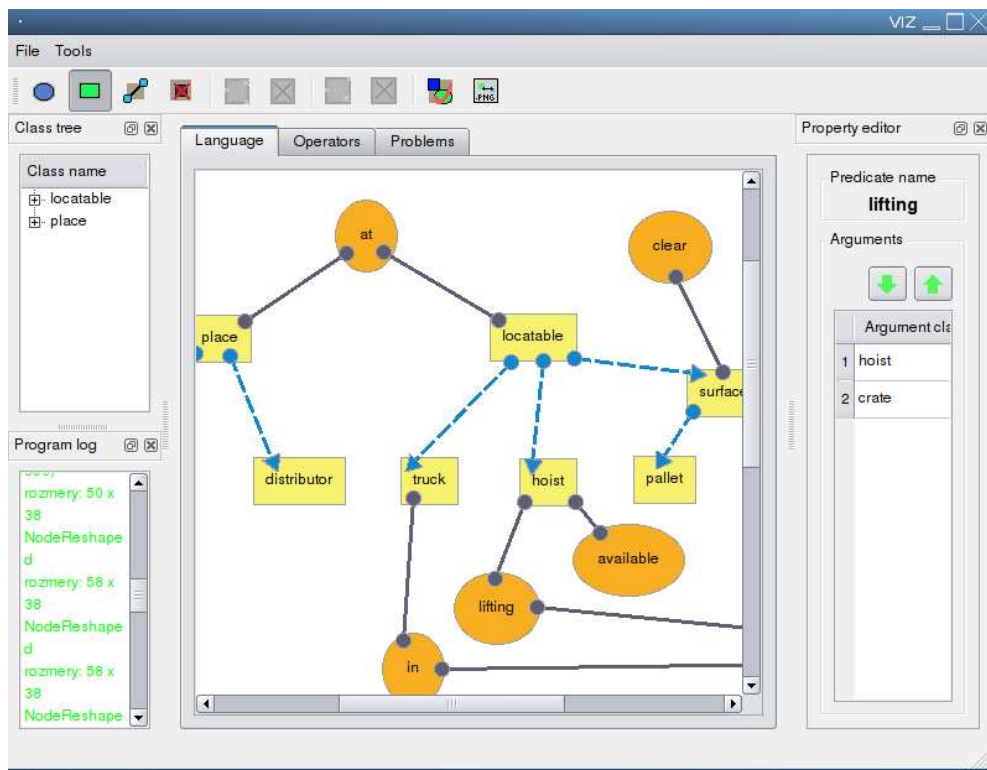
- `extends`
- `constants`
- `timeless`

Dále v sekci `requirements` připouští pouze flagy `:strips` a `:typing`.

5.2 Popis rozhraní

Hlavní okno programu je rozděleno na několik částí (obr. 5.1). Popis hlavního menu a nástrojové lišty je uveden v 5.2.1 a 5.2.2. Zde je uveden popis ostatních částí.

Editační plochy jsou klíčovou součástí uživatelského rozhraní. Je možné mezi nimi přepínat pomocí záložek `Language`, `Operators` a `Problems`.



Obrázek 5.1: Hlavní okno

Umožňují jednoduše navrhnout plánovací doménu pomocí myši, nebo jiného polohovacího zařízení. Klávesnice je při návrhu třeba pouze pro zadání textových hodnot jako např. jména tříd a predikátů, názvy operátorů, nebo názvy plánovacích problémů. Je-li třeba zadat textový vstup, je k tomuto účelu vyvoláno jednoduché dialogové okno. Vstupní řetězce musí začínat písmenem a skládají se¹ z písmen, číslic a znaků "-" a "_". Délka řetězců je omezená a velikost znaků nehraje žádnou roli.

Návrh plánovací domény probíhá postupně ve třech krocích (4.3). Program nabízí editační plochu pro každý z nich. Detailní popis editačních ploch je uveden v 5.2.3.

Pro zobrazení *hierarchie nadefinovaných tříd* slouží dokovací okno:

Class tree

Toto okno nabízí přehled všech tříd, v hierarchickém uspořádání, podle dědičných vztahů mezi nimi.

Pro zobrazení informací o uzlech zobrazených v diagramu, slouží dokovací okno:

Property editor

Vyvolání těchto informací se provádí kliknutím pravého tlačítka myši na zvolený uzel. Charakter vyvolaných informací závisí na kontextu (která z editačních ploch je aktivní) a na typu uzlu (elipsa/obdélník). Význam položek, které jsou pro jednotlivé situace zobrazeny, je popsán v příslušných odílech sekce 5.2.3.

Informace o prováděných akcích a výsledky kontroly diagramů jsou zobrazovány v dokovacím okně:

Program log

Podle výsledků kontroly je možné provádět v příslušných diagramech opravy chyb.

5.2.1 Hlavní menu

V menu **File** jsou především položky pro souborové operace:

- **New** - vyresetuje všechny diagramy a umožní editaci nové domény

¹Dle požadavků jazyka PDDL











- **Open** - umožní otevření souboru s popisem domény v **XML**
- **Save** - umožní uložení domény ve formátu **XML**
- **Properties** - dialog pro nastavení názvu domény, a komentáře
- **Export PDDL**
 - **Export domain** - do zvoleného souboru zapíše **PDDL** kód reprezentující aktuální doménu
 - **Export problems** - do zvoleného adresáře exportuje **PDDL** kód reprezentující plánovací problémy nadefinované pro aktuální doménu. Názvy souborů jsou generovány ve tvaru:

název domény_název problému.pddl

- **Exit** - ukončí program

Menu **Tools** umožňuje vyčištění logovacího okna (**Clear log**), provedení výpisu o stavu domény do logu programu (**Domain summary**) a opětovné vyvolání zavřených dokovacích oken.

5.2.2 Nástrojová lišta

-  přepne program do módu pro přidávání *predikátů*
-  přepne program do módu pro přidávání obdélníkových uzlů
-  přepne program do módu pro přidávání hran
-  přepne program do módu pro mazání
-  vytvoří prázdný diagram pro popis nového *plánovacího operátoru*
-  vymaže z domény aktuálně zobrazený *plánovací operátor*
-  vytvoří prázdný diagram pro popis *plánovacího problému*
-  vymaže aktuální *plánovací problém*
-  provede kontrolu aktuálně zobrazeného diagramu
-  uloží aktuální diagram jako obrázek ve formátu PNG

5.2.3 Popis editačních ploch

Všechny editační plochy jsou ovládány pomocí prvních čtyř tlačítek *nástrojové lišty*. Stisknuté tlačítko určuje, jakým způsobem bude daná plocha reagovat. Způsob reakce zůstává stejný i při přepnutí na jinou editační plochu.

Přidání *tříd*, *proměnných* a *objektů*:

- přepneme program do módu pro přidávání obdélníkových uzlů:



- klikneme **levým** tlačítkem na editační plochu

Podle typu editační plochy bude vyvolán dialog, který je třeba vyplnit pro přidání uzlu. Typ uzlu závisí na editační ploše:

- **Language** - přidáný uzel představuje *třidu*
- **Operators** - přidáný uzel představuje *proměnnou*
- **Problems** - přidáný uzel představuje *objekt*

Predikát lze přidat obdobným způsobem - pouze je třeba přepnout program do módu pro přidávání predikátů:



Před přidáním je opět třeba vyplnit dialog, charakteristický pro každou z editačních ploch.

Chceme-li přidat hranu, je nutné přepnout program příslušného módu pomocí tlačítka:



Hranu lze přidat tak, že stiskem levého tlačítka definujeme její počátek a uvolněním konec. Během této operace bude hrana naznačena čarou. Konce této čáry musí zasahovat do plochy dvou různých uzlů (obr. 5.2), jinak hrana nebude přidána.

Součástí diagramu (uzly/hrany) lze mazat takto:

- přepneme program do módu pro mazání



- klikneme **levým** tlačítkem myši na uzel (hranu), který (kterou) chceme smazat

Následuje popis odlišností jednotlivých editačních ploch.

Language - tato editační plocha slouží k definici *tříd* a *predikátů* pro potřeby plánovací domény. Realizuje se zde první krok návrhu plánovací domény (viz. 4.3).

Dialogy pro přidávání uzlů na této editační ploše vyžadují pouze zadání názvu (obr. 5.3).

V případě *třídy* musí být název unikátní. Název **object** je vyhrazen - v programu představuje společného předka pro všechny ostatní třídy.

Jméno nově přidaného *predikátu* nemusí být unikátní - program umožňuje definovat přetížené predikáty. Pro použití v ostatních diagramech ale musí mít všechny definice přetíženého predikátu stejnou aritu, jinak predikát nebude dostupný k dalšímu použití.

Na této editační ploše lze přidávat *hrany* obou typů (obr. 4.1). Typ hrany je přitom automaticky odvozen dle typů propojovaných uzlů.

Orientované hrany, definující dědičnost, směřují od *předka* k *potomkovi*. Neorientované hrany definují argumenty predikátu. Přehled nadefinovaných dědičných vztahů je k dispozici v okně:



Class tree

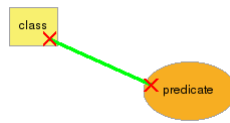
Při kliknutí **pravého** tlačítka myši na *třídu* je zobrazeno v okně:

Property editor

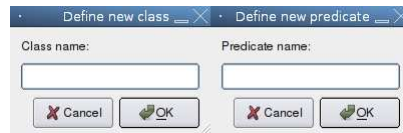
pouze jméno této třídy. Pokud ale klikneme pravým tlačítkem na *predikát*, je zobrazen přehled připojených argumentů (obr. 5.4). Označíme-li položku v tabulce argumentů, můžeme šipkami měnit její pořadí.

Operators - po přepnutí na tuto editační plochu máme možnost editovat *plánovací operátory*.

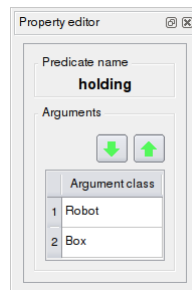
- pomocí tlačítka  lze přidat nový operátor
- tlačítkem  smažeme aktuální operátor
- pro aktivaci editační plochy je třeba přidat alespoň jeden operátor
- mezi existujícími operátory se můžeme přepínat pomocí boxu nad editační plochou (obr. 5.5)



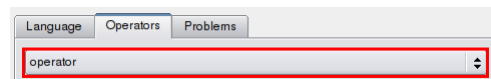
Obrázek 5.2: Přidání nové hrany



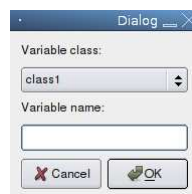
Obrázek 5.3: **Language**: přidání nového uzlu



Obrázek 5.4: **Language** : informace o predikátu



Obrázek 5.5: **Operators**: přepínání operátorů



Obrázek 5.6: **Operators**: přidání proměnné

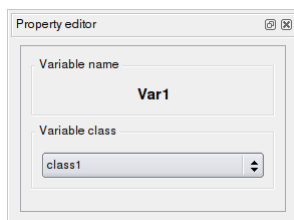
Dialog pro přidání nové *proměnné* je na obrázku 5.6. Je v něm třeba zvolit *třídu*, která určí typ proměnné. Poté je nutné specifikovat pro proměnnou unikátní² jméno.

Při přidávání *predikátu* je třeba vyplnit dialog 5.7. Tento dialog obsahuje nabídku definovaných predikátů a lze v něm zaškrtnout, do které **množiny** v rámci daného *plánovacího operátoru* patří³.



Obrázek 5.7: **Operators:** přidání predikátu



Po kliknutí pravého tlačítka myši na *proměnnou* je zobrazeno v **Property editoru** jméno této proměnné. Především je zde ale možnost zvolit pro tuto proměnnou jinou třídu (obr. 5.8).



Obrázek 5.8: **Operators:** informace o proměnné

Kliknutím pravého tlačítka myši na predikát také změníme obsah **Property editoru**. Tentokrát dostaneme možnost změnit množinu daného predikátu a pořadí jeho parametrů, jak je vidět na obrázku 5.9.

Problems - tato editační plocha slouží k definici *plánovacích problémů*.

- pomocí tlačítka  můžeme vytvořit nový problém
- tlačítko  slouží k vymazání aktuálního problému

²v rámci operátoru

³viz. definice plánovacího operátoru v 2.1.1

- pomocí boxu nad editační plochou se můžeme přepínat mezi existujícími plánovacími problémy (obr. 5.10)
- pomocí zaškrtačích tlačítek lze skrýt/zobrazit zvolenou množinu predikátů
 - `Init` odpovídá množině s_0
 - `Goal` odpovídá množině g

Dialog pro přidávání *objektů* opět požaduje specifikaci *třídy* a unikátního názvu objektu. Dialogové okno vypadá stejně jako na obr. 5.6 - pouze místo slova `Variable` je v něm uvedeno `Object`.

Pro přidání *predikátu* je třeba v dialogu (obr. 5.11) vybrat o jaký predikát se jedná a zvolit množinu, do které v rámci daného plánovacího problému náleží.

Po kliknutí pravého tlačítka myši na *objekt* je zobrazeno v `Property editoru` jméno objektu. Můžeme zde rovněž změnit třídu objektu. Situace vypadá stejně jako na obr. 5.8 - pouze místo slova `Variable` je uvedeno `Object`.

Po kliknutí pravého tlačítka myši na *predikát* můžeme v `Property editoru` změnit množinu, do které tento predikát v rámci daného plánovacího problému patří (obr. 5.12). Můžeme zde také změnit pořadí argumentů predikátu.

5.3 Demontrace

V této části jsou předvedeny funkce programu na příkladu. Je zde popsána definice jednoduché plánovací domény pro *Svět kostek* [8].


5.3.1 Svět kostek

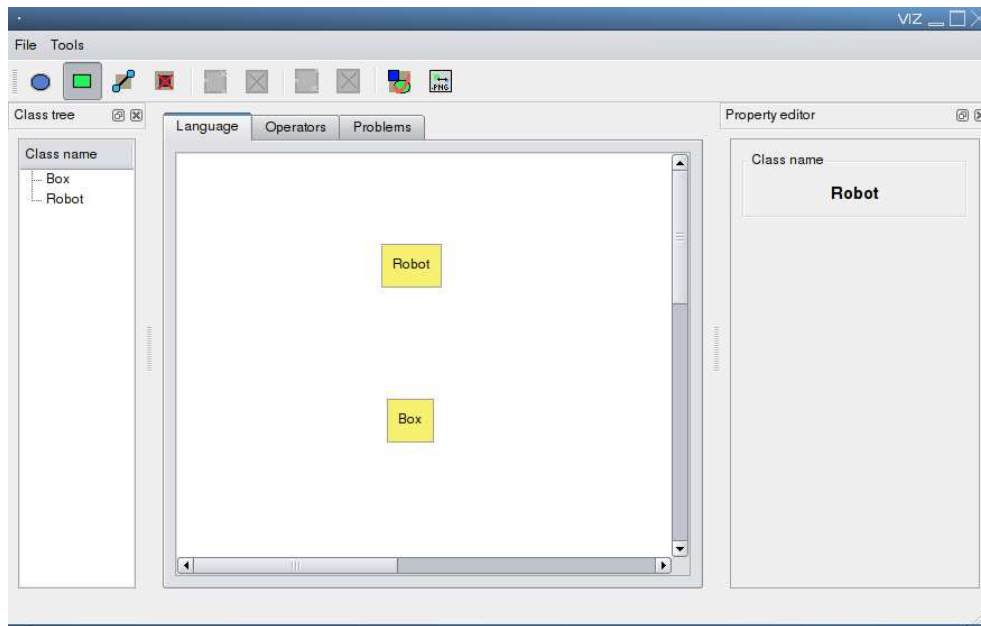
Svět kostek je prostředí, které obsahuje jednoho robota a kostky. Robot zde představuje agenta, který kostky přerovnává. Kostky mohou ležet buď přímo na stole, nebo na sobě. Přitom předpokládáme, že kostka nemůže ležet na více kostkách současně. Robot dokáže kostky od sebe rozlišit (dejme tomu, že jsou označeny písmenky) a dovede libovolnou z nich zvednout a položit na libovolnou přípustnou pozici (tj. kamkoliv na zem, nebo na jinou kostku, na které žádná jiná kostka už neleží).

Podobný popis je možné k plánovací doméně připojit pomocí


File -> Properties

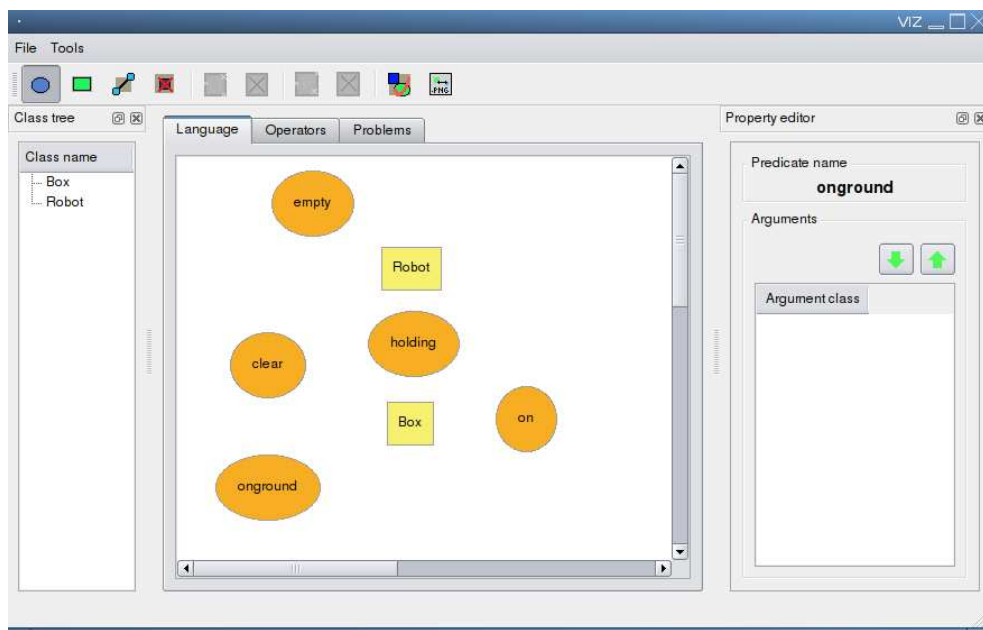
5.3.2 Definice jazyka


Svět kostek obsahuje pouze dva typy objektů: kostky a robota. Je-li aktivní záložka **Language**, přepneme program do správného módu tlačítkem  a pro každý typ nadefinujeme třídu:

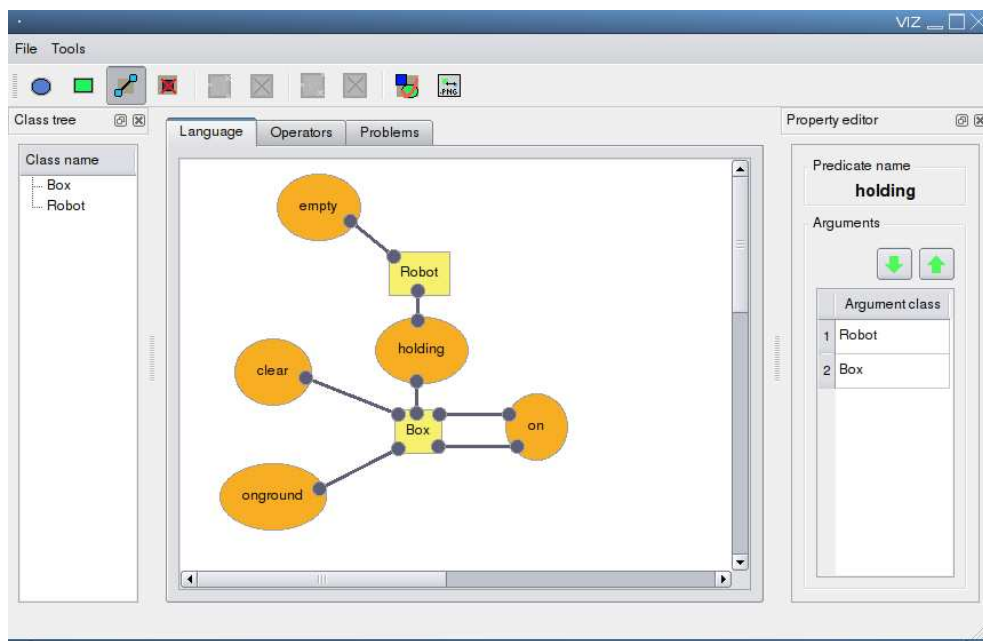


Dalším krokem je definice predikátů. Při návrhu plánovací domény je definice predikátových symbolů jeden z nejdůležitějších kroků. Predikáty nám později umožní popsat stav prostředí.

Přepneme tedy program do požadovaného módu tlačítkem  a definujeme predikátové symboly:



Argumenty predikátů definujeme tak, že propojíme uzly reprezentující příslušné predikáty se třídami. Každá hrana má tak význam jednoho argumentu, jehož typ je určen třídou, do které vede. Přepneme program do příslušného módu pomocí tlačítka  a určíme argumenty nadefinovaných predikátů:




Odpovídající kód domény v PDDL je:

; Svět kostek je prostředí, které obsahuje jednoho robota a kostky.
 ; Robot zde představuje agenta, který kostky přerovnává.




```
(define (domain boxworld )
  (:requirements :strips :typing)
  (:types
    Box Robot - object
  )
  (:predicates
    (empty ?a - Robot)
    (clear ?a - Box)
    (onground ?a - Box)
    (holding ?a - Robot ?b - Box)
    (on ?a - Box ?b - Box)
  )
)
```

5.3.3 Definice operátorů

V této části popíšeme definici jednoho plánovacího operátoru. Uvažujme například operátor `unstack` pro zvednutí kostky z vrcholu nějaké věže kostek.

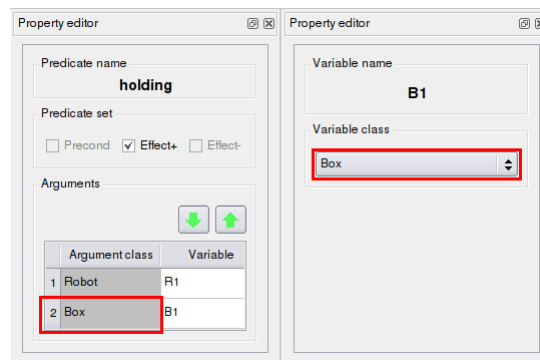
Nejprve se přepneme do záložky **Operators**. Tím se aktivuje tlačítko , které využijeme pro přidání prázdného diagramu pro nový operátor:



Poté postupně přidáme proměnné () , predikáty () a pomocí hran () určíme argumenty.

Při definici predikátů určíme, jestli jde o *prekondici*, *pozitivní efekt*, nebo *negativní efekt* zaškrtnutím příslušných políček v dialogu (obr. 5.7).

- Pokud nejde přidat hrana, je možné, že typ proměnné a typ argumentu u predikátu nejsou kompatibilní. Typ proměnné přitom může být libovolný potomek třídy uvedené jako typ argumentu, nebo přímo tato třída.

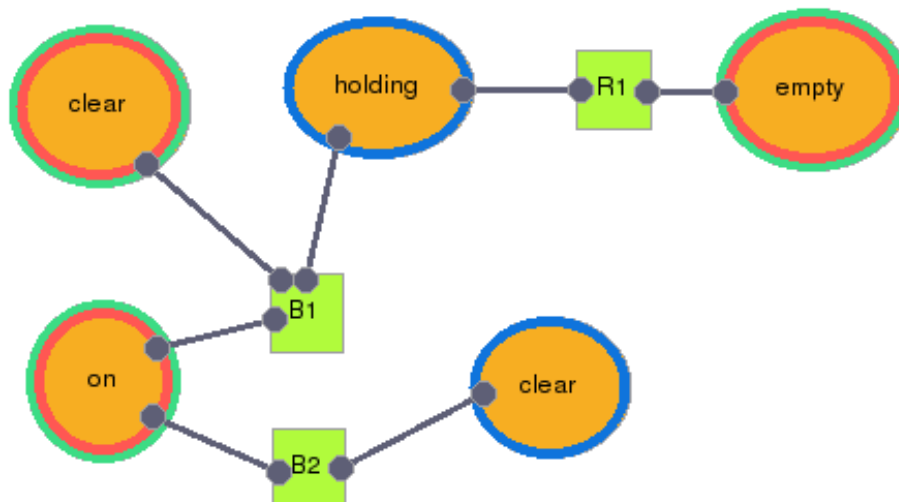


- Chceme-li zajistit, aby v predikátu `on(?a - Box ?b - Box)` první argument odpovídal „vrchní“ kostce, můžeme to provést v **Property editoru**:
 - klikneme pravým tlačítkem na predikát `on`
 - označíme chybně dosazenou proměnnou

- pomocí šipek změníme pořadí argumentů predikátu

	Argument class	Variable
1	Box	B1
2	Box	B2

Výsledný diagram:



Je převeden do PDDL jako:

```
(:action unstack
  :parameters (?R1 - Robot ?B1 - Box ?B2 - Box )
  :precondition (and
    (on ?B1 ?B2)
    (empty ?R1)
    (clear ?B1)
```

```


)
:effect (and
  (clear ?B2)
  (holding ?R1 ?B1)
  (not (on ?B1 ?B2))
  (not (empty ?R1))
  (not (clear ?B1))
)
)

```

Ostatní operátory nadefinujeme obdobným způsobem.




5.3.4 Definice plánovacího problému

V této části je popsána definice plánovacího problému.

Nejprve je třeba aktivovat záložku **Problems**. Poté pomocí tlačítka  vytvoříme prázdný diagram pro definici nového plánovacího problému:

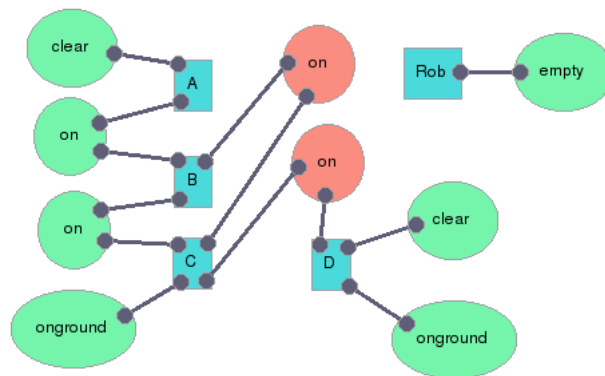


Název, který zde uvedeme, slouží především pro orientaci mezi definovanými problémy. Mimo to je také součástí jména souboru s daným problémem při exportu do PDDL.

Postup vytváření diagramu je obdobný jako v předchozích případech. Nejprve přidáme objekty () , poté predikáty () a pomocí hran () určíme argumenty.

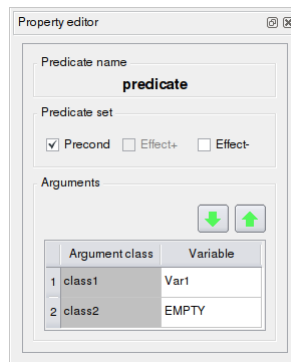
V dialogu pro přidávání predikátů je možné specifikovat množinu plánovacího problému, do které bude predikát přidán (obr. 5.11). Podle zvolené možnosti jsou predikáty obarveny buď zelenou (**Init**), nebo červenou (**Goal**) barvou. Pomocí **Property editoru** lze množinu změnit (5.12).

Hotový problém může vypadat například takto:

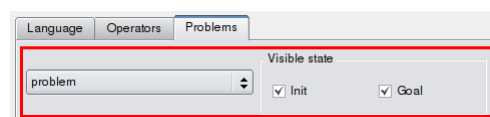


Po exportu plánovacieho problému do PDDL obdržíme kód:

```
(define (problem fourBoxes)
  (:domain boxworld )
  (:objects
    D C B A - Box
    Rob - Robot
  )
  (:init
    (empty Rob)
    (clear A)
    (on A B)
    (on B C)
    (onground C)
    (onground D)
    (clear D)
  )
  (:goal
    (and
      (on B C)
      (on C D)
    )
  )
)
```



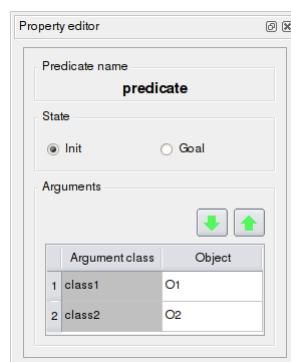
Obrázek 5.9: **Operators**: informace o predikátu



Obrázek 5.10: **Problems**: přepínání problémů



Obrázek 5.11: **Problems**: přidání predikátu



Obrázek 5.12: **Problems**: informace o predikátu

Kapitola 6

Programátorská dokumentace

6.1 Jazyk, knihovny a prostředí

Implementace programu je provedena v jazyce C++ s použitím knihoven:

- Qt GUI Toolkit - Open Source Edition [9]
- igraph [10]

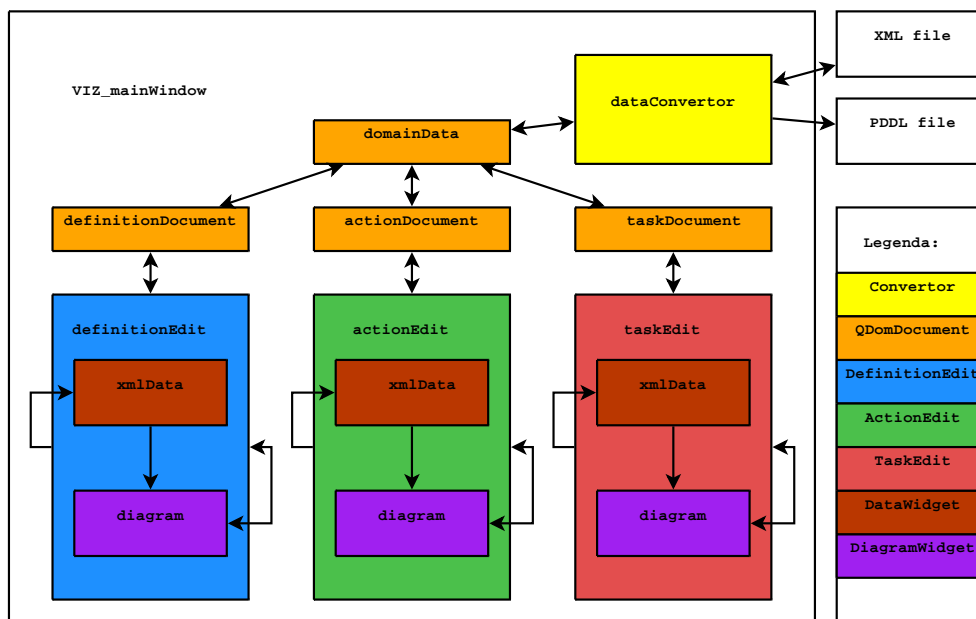
Pro vytvoření GUI byl použit program Qt Designer.

Knihovna Qt umožňuje vytvářet aplikace z tzv. *widgetů*. Nabízí také širokou škálu základních tříd, které lze s výhodou využít při tvorbě nových widgetů. Pro implementaci tohoto programu byla vytvořena speciální sada widgetů, které umožňují vizualizaci a editaci diagramů, reprezentujících plánovací doménu. Vývoj programu byl proveden v operačním systému:

Debian Linux (kernel 2.6.26)

Přenositelnost celého programu na jinou platformu závisí na přenositelnosti použitých knihoven. Kompilace programu by měla bez problémů proběhnout v linuxovém prostředí, které splňuje následující podmínky:

- verze překladače gcc 4.3.2
- verze toolkitu Qt 4.4.3
- verze knihovny igraph 0.5.2



Obrázek 6.1: Schéma hlavní třídy programu

6.2 Architektura programu

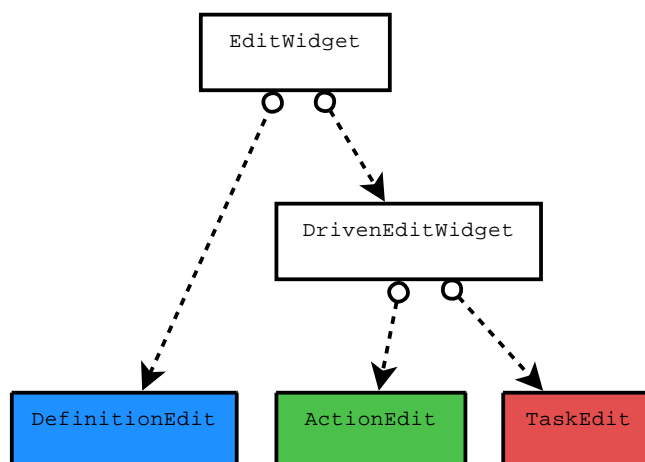
Schéma hlavní třídy `VIZ_mainWindow` je naznačeno na obrázku 6.1. Komponenty jsou barevně rozlišeny podle svých tříd. Mimo třídu `QDomDocument`, která je součástí knihovny `Qt`, je v této sekci popsán význam všech zobrazených tříd. Šipky ve schématu značí vybraná rozhraní mezi jednotlivými komponentami. Označení tříd a komponent je totožné s označením ve zdrojovém kódu ¹.

6.2.1 Convertor

Tato třída zajišťuje vstup a výstup programu z/do souboru. V případě XML se jedná o vstup i výstup, v případě PDDL je možný pouze výstup. Rozhraní třídy tvoří funkce:

```
readFromXML
writeToXML
```

¹pouze **definitionEdit**, **actionEdit** a **taskEdit** jsou zde uvedeny bez prefixu **ui**. (jsou přímou součástí uživatelského interface)



Obrázek 6.2: Odvození z třídy **EditWidget**

```

writeDomainToPDDL
writeProblemsToPDDL

```

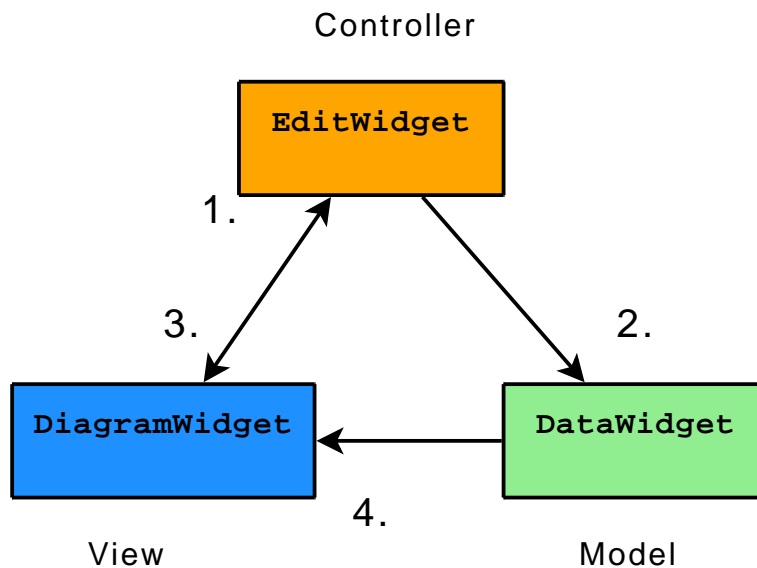
Sestavení PDDL obsahu probíhá postupnou extrakcí potřebných informací z poskytnutého objektu třídy `QDomDocument`. Při načítání dat probíhá pouze kontrola syntaxe XML poskytnutého dokumentu. XML struktura, kterou program používá je popsána formou BNF v dodatku (A). Kontrola sémantiky dokumentů dle této struktury ale není implementována.

6.2.2 EditWidget

Třída `EditWidget` je odvozena ze třídy `QGraphicsView` poskytované knihovnou `Qt`. Jsou z ní odvozeny třídy `DiagramEdit`, `ActionEdit` a `TaskEdit` (obr. 6.2). Tyto třídy jsou klíčovou součástí uživatelského rozhraní.

Objekty těchto tříd (`definitionEdit`, `actionEdit` a `taskEdit` na obrázku 6.1) zajišťují postupně provedení všech kroků, které jsou třeba k návrhu plánovací domény (4.3):

1. `definitionEdit` - deklarace *tříd* a *predikátových symbolů*
2. `actionEdit` - definice *plánovacího operátoru*
3. `taskEdit` - definice *plánovacího problému*



Obrázek 6.3: MVC schema

Trojice tříd `EditWidget`, `DataWidget` a `DiagramWidget` tvoří dohromady jádro programu. Vztahy mezi nimi jsou navrženy podle schématu *Model-view-controller* [11].

Na obrázku 6.3 je toto schéma zobrazeno s příslušnými třídami. Šipky na obrázku reprezentují rozhraní mezi propojenými třídami. Pro snadnější orientaci jsou očíslovány - čísla slouží v následujícím textu jako reference.

Rozhraní třídy `EditWidget` vůči třídě `DiagramWidget` (č. 1.) tvoří sada *událostí*, které třída `EditWidget` zpracovává ve funkci `EditWidget::event`. Program využívá mechanismus pro zpracování událostí, poskytovaný knihovnou Qt. V souboru `CustomEvents.h` je k tomuto účelu definována třída `DiagramEvent`, která rozšiřuje třídu `QEvent` o možnost přenášet dodatečné informace. Tyto informace sestávají z číselného identifikátoru objektu² přístupného pomocí metody:

```
int DiagramEvent::itemID()
```

a upřesňujících dat, která lze získat voláním:

```
QVariant DiagramEvent::data()
```

Stručný popis událostí je uveden v tabulce 6.2.2.

²hrana/uzel

název události	slovní popis	upřesňující data
NodeDrag	uzel je přesouván	vektor
NodeMoved	konec přesunu uzlu	vektor
EdgeMoved	konec přesunu hrany	specifikace konce, vektor
EdgeDefined	byla definována hrana	ID počátku, ID konce pozice počátku pozice konce
NodeReshaped	byl změněn tvar uzlu	-
NodeLeftClick	smazání uzlu (dle aktuálního módu)	-
NodeRightClick	požadavek na detailní informace	-
EdgeLeftClick	smazání hrany (dle aktuálního módu)	-
EdgeRightClick	nevyužitá událost	-
DiagramLeftClick	byl definován uzel (vygeneruje se unikátní identifikátor)	pozice
DiagramRightClick	nevyužitá událost	-

Tabulka 6.1: Události

název metody	popis
<code>addDataNode</code>	přidá nový uzel
<code>addDataEdge</code>	přidá novou hranu
<code>delDataNode</code>	smaže uzel
<code>delDataEdge</code>	smaže hranu
<code>changeNodePos</code>	přepíše souřadnice uzlu
<code>changeEdgePos</code>	přepíše souřadnice hrany
<code>changeEdgeAssociation</code>	přepíše id uzlu, se kterým je hrana spojena
<code>connectEdgeToNode</code>	přidá do seznamu spojení cílového uzlu id dané hrany
<code>disconnectEdgeFromNode</code>	odebere ze seznamu spojení cílového uzlu id dané hrany

Tabulka 6.2: Rozhraní **DataWidget-EditWidget**

6.2.3 DataWidget

Třída `DataWidget` obstarává manipulaci s daty. Její obsah je inicializován voláním metody:

```
void DataWidget::loadData(QDomDocument diagramDocument)
```

Protože třída `QDomDocument` používá explicitní sdílení dat, jsou změny prováděny přímo v dokumentech `definitionDocument`, `actionDocument` nebo `taskDocument` (záleží na kontextu - viz. 6.1).

Tabulka 6.2 uvádí nejdůležitější metody rozhraní třídy `DataWidget` vůči třídě `EditWidget` (č. 2. z obrázku 6.3).

Změny jako přidání, nebo odebrání uzlu/hrany je třeba propagovat do příslušné instance třídy `DiagramWidget`. Ukazatel na ni je uložen v privátní proměnné `DataWidget::diagramScene`. Tím je umožněno volání funkcí, které poskytuje třída `DiagramWidget` v rozhraní č. 4. (obr. 6.3).

6.2.4 DiagramWidget

Třída `DiagramWidget` je odvozena ze třídy `QGraphicsScene`. V programu zajišťuje spolu se třídou `EditWidget` vizualizaci diagramů. Objekty, které zobrazuje jsou instancemi tříd:

- `Node` v souboru `Node.h`

název metody	popis
<code>setDiagramMode</code>	nastaví mód
<code>setNodeVisible</code>	skryje/odkryje uzel
<code>setEdgeVisible</code>	skryje/odkryje hranu
<code>translateEdge</code>	posune počátek/konec hrany o vektor
<code>stickEdgeToNode</code>	„přitáhne“ počátek/konec hrany k danému uzlu ³
<code>collidingNodeID</code>	zjistí ID uzlu, kolidujícího s počátkem/koncem hrany

Tabulka 6.3: Rozhraní **DiagramWidget-EditWidget**

- `Edge` v souboru `Edge.h`

Tyto objekty jsou označeny *číselným identifikátorem*. Identifikátor je unikátní v rámci diagramu. Ukazatele na zmíněné objekty `DiagramWidget` jsou uloženy v kontejnerech typu `QHash`:

- `QHash<int,Node*> DiagramWidget::dgwNodes`
- `QHash<int,Edge*> DiagramWidget::dgwEdges`

Klíčem pro přístup k prvkům těchto kontejnerů jsou identifikátory příslušných objektů.

- metody tvořící rozhraní třídy `DiagramWidget` vůči třídě `EditWidget` (č. 3. na obrázku 6.3) jsou uvedeny v tabulce 6.3. Mód nastavený metodou `DiagramWidget::setDiagramMode` ovlivňuje zpracování událostí
- metody tvořící rozhraní třídy `DiagramWidget` vůči třídě `DataWidget` (č. 4. na obrázku 6.3) jsou uvedeny v tabulce 6.4.
Metoda `DiagramWidget::updateNode` je volána při změně predikátů (změna barvy `init/goal`, nebo změna obtažení `precond/effect+/effect-`)

název metody	popis
<code>addNode</code>	přidá nový uzel
<code>addEdge</code>	přidá novou hranu
<code>removeNode</code>	odebere uzel
<code>removeEdge</code>	odebere hranu
<code>updateNode</code>	provede aktualizaci uzlu

Tabulka 6.4: Rozhraní **DiagramWidget-DataWidget**

Kapitola 7

Závěr

V rámci práce byl popsán přehledný způsob, který umožňuje popsat jednoduché plánovací domény pomocí diagramů. Jeho použitelnost byla doložena implementací programu, ve kterém je možné pomocí těchto diagramů realizovat návrh plánovacích domén.

Uživatelské rozhraní programu se snaží poskytnout uživateli co největší volnost při návrhu plánovacích domén (n -ární predikátové symboly, přetěžování predikátových symbolů) a zároveň je navrženo tak, aby pomáhalo předcházet chybám (typová kontrola argumentů, kontrola dědičnosti tříd).

Práce s programem nevyžaduje hlubší znalosti jazyka PDDL. S jeho použitím tak odpadá nutnost editace nepřehledných textových souborů a pro uživatele představuje snadněji pochopitelnou alternativu k doposud užívaným programům.

Program umožňuje vytváření nových plánovacích domén a umožňuje jejich export do jazyka PDDL. Vygenerované soubory lze použít jako vstup pro externí plánovač. Testy byly prováděny s plánovači: **SGPlan** a **SatPlan**.

Import z PDDL bohužel není podporován. Pro implementaci nestačí pouze převod z PDDL do XML - je také třeba vygenerovat informace o umístění uzlů na ploše. Samotný převod představuje vzhledem k rozdílnému charakteru obou formátů značnou komplikaci.

Způsob, který je zde pro vizualizaci plánovacích domén využit, umožňuje rozšíření o další vyjadřovací prostředky jazyka PDDL. Do budoucna by mohly být možnosti programu rozšířeny například o práci s konstantami, nebo funkčními symboly.

Dále je možné přidat možnost spolupráce s externími plánovači. Například za účelem vizualizace plánů.

Dodatek A

XML struktura

Zde je uveden popis XML struktury využívané programem.

- Syntaktické prvky jsou označeny velkým písmem (např. DOMAIN)
- Expanze prvků je naznačena jako ::=
- Nepovinné prvky jsou uzavřeny do hranatých závorek (např. CONNECTION_LIST)
- Opakování prvků je naznačeno znakem * (0 a více) nebo + (1 a více)
- Znakem | jsou odděleny alternativy pro daný prvek (např. precondition | effect+ | effect-)

```
DOMAIN ::= <domain>
DEFINITION
OPERATORS
PROBLEMS
</domain>
```

```
DEFINITION ::= <definition>
DIAGRAM
</definition>
```

```
OPERATORS ::= <operators>
ACTION_LIST
</operators>
```

```

PROBLEMS ::= <problems>
TASK_LIST
</problems>

ACTION_LIST ::= ACTION *

ACTION ::= <action name="STRING" >
DIAGRAM
</action>

TASK_LIST ::= TASK *

TASK ::= <task name="STRING" >
DIAGRAM
</task>

DIAGRAM ::= <diagram>
NODE_LIST
EDGE_LIST
</diagram>

NODE_LIST ::= NODE *

EDGE_LIST ::= EDGE *

NODE ::= <node id="INTEGER" >
<label>STRING</label>
<type>NODE_TYPE</type>
[PREDICATE_SET]
POSITION
[CONNECTION_LIST]
</node>

NODE_TYPE ::= class | variable | object | predicate

PREDICATE_SET ::= SET_DESC *

```

```

SET_DESC ::= <set>SET_LABEL</set>

SET_LABEL ::= precondition | effect+ | effect-

POSITION ::= <pos>
<x>INTEGER</x>
<y>INTEGER</y>
</pos>

CONNECTION_LIST ::= <connections>
CONNECTION +
</connections>

CONNECTION ::= STARTING | ENDING

STARTING ::= <starts [argn="INTEGER"] >INTEGER</starts>

ENDING ::= <ends>INTEGER</ends>

EDGE ::= <edge id="INTEGER" >
<purpose>EDGE_PURPOSE</purpose>
<start nid="INTEGER">
POSITION
</start>
<end nid="INTEGER">
POSITION
</end>
</edge>

EDGE_PURPOSE ::= association | inheritance

INTEGER ::= number value

STRING ::= string value

```


Dodatek B

Obsah přiloženého CD

Následují jména adresářů s obsahem:

- **document** - obsahuje elektronickou verzi tohoto dokumentu (v podadresáři **tex** jsou zdrojové kódy v $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}_{\text{U}}$)
- **library** - obsahuje knihovny potřebné pro kompilaci programu
- **source** - obsahuje zdrojové kódy programu + soubor **README** s instrukcemi pro kompilaci
- **XML_examples** - obsahuje příklady plánovacích domén vytvořené s použitím programu

Literatura

- [1] Russell S, Norvig P (1995): *Artificial Intelligence: A Modern Approach*, Prentice Hall Series in Artificial Intelligence. Englewood Cliffs, New Jersey
- [2] Ghallab M., Nau D., Traverso P.: *Automated Planning, Theory and Practice*, Elsevier, Morgan Kaufmann Publishers 2004
- [3] Chih-Wei Hsu, Benjamin W. Wah, Ruoyun Huang, and Yixin Chen: *SGPlan 5: Subgoal Partitioning and Resolution in Planning*
<http://manip.crhc.uiuc.edu/programs/SGPlan/index.html>
- [4] Henry Kautz, Bart Selman, and Joerg Hoffmann: *SatPlan: Planning as Satisfiability* Abstracts of the 5th International Planning Competition, 2006.
- [5] Ron M. Simpson: *GIPO graphical interface for planning with objects*
<http://scom.hud.ac.uk/planform/gipo/>
- [6] Tiago Stegun Vaquero, Flavio Tonidandel, José Reinaldo Silva: *The itSIMPLE tool for modeling planning domains*
<http://dlab.poli.usp.br/twiki/bin/view/ItSIMPLE/WebHome>
- [7] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins: *PDDL - the planning domain definition language*. Technical report, Yale Center for Computational Vision and Control, 1998.
- [8] John K. Slaney, Sylvie Thiébaux: *Blocks World revisited*. Artif. Intell. (AI) 125(1-2):119-153 (2001)
- [9] Dokumentace knihovny Qt 4.4: <http://doc.trolltech.com/4.4/index.htm>

- [10] Gábor Csárdi, Tamás Nepusz: *The igraph software package for complex network research*. InterJournal Complex Systems, 1695, 2006.
- [11] <http://en.wikipedia.org/wiki/Model-view-controller>