VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

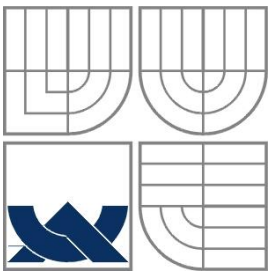
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

SIMULACE A NÁVRH INTELIGENTNÍCH AGENTŮ

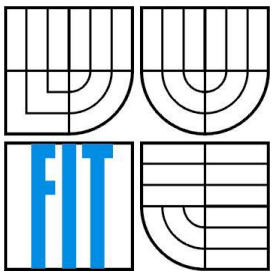
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. SVATOPLUK ŠPERKA
AUTHOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

SIMULACE A NÁVRH INTELIGENTNÍCH AGENTŮ

SIMULATION AND DESIGN OF INTELIGENT AGENTS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. SVATOPLUK ŠPERKA

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. VLADIMÍR JANOUŠEK, Ph.D.

BRNO 2009

Zadání

1. Seznamte se s přístupy k návrhu inteligentních agentů, a to jak reaktivních, tak deliberativních. Podrobněji prostudujte subsumpční architekturu a model založený na BDI teorii.
2. Seznamte se s existujícími modely agentů a agentních architektur na bázi DEVS. Seznamte se i s podpůrnými prostředky pro návrh agentů na bázi simulace.
3. Sestavte či vyberte vhodné testovací úlohy a ty v daných prostředích implementujte.
4. Porovnejte implementace a výsledky simulací testovacích úloh na základě různých kritérií.
5. Na základě výsledků zhodnoťte vhodnost platforem pro modelování kontrolérů fyzických agentů (robotů).
6. Navrhněte možné úpravy a modifikace existujících prostředků pro modelování a simulaci agentů v prostředí PNtalk-SmallDEVS.

Abstrakt

Konvenční způsob vývoje opakující fáze návrhu, implementace a testování není adekvátní pro systémy třídy inteligentních agentů, u nichž je vyžadováno komplexní chování, ale jejich specifikace je na počátku nejasná. Jako vhodnější se jeví inkrementální tvorba modelu agenta v simulaci, která dává návrháři přímou zpětnou vazbu v podobě změn chování systému. Tato interaktivita nejenže urychluje vývoj, ale také, díky novým znalostem o chování získaných během simulace, zpřístupňuje návrháři nové části prostoru možných modelů. Tato práce se zabývá srovnáním vhodnosti dvou přístupů k návrhu inteligentních agentů, a s nimi souvisejících platforem v kontextu modelovacího a simulačního frameworku SmallDEVS, pro tento způsob vývoje. Prvním přístupem je reaktivní subsumpční architektura založená na formalismu DEVS a druhým framework PNagent realizující deliberativní BDI architekturu pomocí Objektově Orientovaných Petriho Sítí.

Klíčová slova

agent, robot, subsumpční architektura, BDI, metodologie vývoje, model-based design, kontinuita modelu, modelování, simulace, PNagent, PNtalk, DEVS, SmallDEVS

Abstract

Conventional method of development which repeats phases of design, implementation and testing is not adequate for systems like intelligent agents for which complex behavior is required but specification is unclear at the beginning of development process. Incremental design of agent's model during simulation seems more suitable for it enables direct feedback in behavioral changes of a system. This interactivity speeds up development process and helps to uncover parts of a space of all models to designer – thanks to new knowledge acquired during simulation. This thesis aims to provide comparison of suitability of two agent architectures and respective platforms in context of SmallDEVS modeling and simulation framework for this methodology of development. First approach is reactive and decentralized subsumption architecture based on DEVS formalism and the second one is PNagent, framework realizing deliberative BDI architecture using Object Oriented Petri Nets.

Keywords

agent, robot, subsumption architecture, BDI, development methodology, model-based design, model continuity, modeling, simulation, PNagent, PNtalk, DEVS, SmallDEVS

Citace

Svatopluk Šperka: Simulace a návrh inteligentních agentů. Brno, 2009, diplomová práce, FIT VUT v Brně

Návrh a simulace inteligentních agentů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Vladimíra Janouška, Ph.D.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Svatopluk Šperka
26.5.2009

Poděkování

Tímto bych chtěl poděkovat svému vedoucímu Ing. Vladimíru Janouškovi, Ph.D. za rady, podnětné nápady a konzultace. Tátovi, který trpělivě věnoval svůj čas provádění korektur průběžných verzí textu. A celé mé rodině a přátelům, kteří trpělivě snášeli mé naříkání a podporovali mě v další práci.

1 Úvod

Vývoj agentů, jakožto softwaru schopného jednat v prostředí za účelem splnění svého cíle, je, stejně jako vývoj jiných složitých systémů, komplikovaný proces. Zvláště pokud se jedná o agenty v reálném světě – roboty – a pokud je snaha vetknout jim prvky inteligence a autonomie. Cílem vývojáře je totiž obdařit agenty dostatečně komplexním, robustním a třeba i pro něj nepředvídatelným chováním, díky kterému se budou moci vyrovnat s nástrahami prostředí, ve kterém se nacházejí.

Klasický model vývoje postavený na oddělených fázích návrhu (zahrnující implementaci) a testování, se nezdá pro tuto třídu systémů adekvátní, protože nedává vývojáři do rukou dostatečnou flexibilitu pro hledání možných řešení, což může neúměrně prodloužit dobu vývoje. Jako vhodnější se jeví metodologie inkrementálního vývoje modelu agenta v simulaci, důsledkem níž je přímá interakce návrháře s modelem v procesu vývoje. Tento přístup umožňuje efektivně realizovat budování modelu zdola-nahoru, který je obecně vhodnější u systémů, u nichž známe cílové chování, ale na začátku neexistuje jasná představa, jakými prostředky ho dosáhnout. Model běžící v simulaci, do kterého jsou prováděny úpravy, poskytuje přímou zpětnou vazbu a tedy další znalosti o jeho chování potenciálně vedoucí k odhalení nové části prostoru všech modelů a tedy k neznámým možným řešením s výhodnějšími vlastnostmi.

U konvenčního modelu vývoje bývá předstupněm implementace systému fáze analýzy, jejímž výstupem je jeho model, specifikovaný více či méně formálními prostředky. Při implementaci je ovšem model postupně nevratně transformován do úplně jiné podoby. Zpravidla se jedná o zdrojový kód ve vybraném programovacím jazyce, který musí být dále transformován kompilací do spustitelné podoby. V podobě zdrojového kódu jsou ale některé aspekty původního modelu nutně zanedbány, zatemněny či rozmělněny – z důvodu rozdílné úrovně abstrakce, na které je systém popisován. Tím se ztrácí názornost a možnost vysokoúrovňového pohledu na systém, které jsou nutné při práci se systémem jako celkem. Je tedy zpravidla vynuceno udržování popisu systému na obou úrovních.

Řešením tohoto problému je tzv. *model-based design*[1]. Ten přistupuje k návrhu systému modelováním pomocí skutečně formálních principů, díky nimž jsou tyto modely následně spustitelné (simulovatelné) – k žádné nevratné transformaci zde tedy nedochází. V průběhu celé implementace se lze neustále opírat o aktuální model na všech úrovních abstrakce, což značně usnadňuje návrh komplexních systémů.

Pokud je model postaven na formálním základě, přináší to i další výhodu – je možné aplikovat formální analýzu a verifikaci přímo na tento vysokoúrovňový popis. Díky tomu se zvýší jistota korektního chování systému – v našem případě agenta. Možnost nekorektního chování ovšem zůstává i v případě, že systém projde formální verifikací. V případě robotických systémů totiž verifikace bude probíhat pouze s modelem prostředí, který nutně nepokryje celé spektrum chování prostředí reálného. Proto se jeví jako vhodné zachovat model i pro skutečný provoz systému, místo

[1] Návrh založený na modelech.

jeho překladu do jiné formy. *Kontinuita modelu*[2], jako extrémní varianta model-based designu, tak zpřístupňuje sledování a analýzu fungování systému v intencích, ve kterých byl modelován. Díky formálnímu základu je jednodušší přesně rekonstruovat běh systému než u programovacích jazyků či strojového kódu, což je velice užitečné v případě výskytu chyby, neočekávaného chování či jen k analýze pro účely dalších iterací procesu vývoje.

Účelem práce je využít inkrementální modelování formálními prostředky s využitím simulace k návrhu inteligentních agentů, a to dvěma rozdílnými přístupy. Následně potom srovnat vhodnost a připravenost těchto přístupů a s nimi spojených platforem z hlediska tohoto konceptu vývoje.

Prvním přístupem je modelování agentů jako reaktivních s využitím *subsumpční architektury* a formalismu *DEVS* v podobě modelovacího a simulačního frameworku *SmallDEVS*. Druhým je potom deliberativní *BDI* architektura *PNagent*, realizovaná v jazyce *PNtalk*. Ten je implementací konceptu *Objektově Orientovaných Petriho Sítí*, kombinující přednosti Petriho sítí a objektově orientovaného přístupu.

1.1 Struktura práce

Nejdříve jsou v druhé kapitole rozebrány současné přístupy k návrhu agentů; postupně popíšeme teoretické základy i konkrétní příklady deliberativních architektur, reaktivních agentů a spojení obou přístupů v podobě hybridních architektur.

Třetí kapitola se zabývá možnostmi vývoje agentů v kontextu modelovacího a simulačního frameworku *SmallDEVS* a na něj napojených technologií. Postupně je detailněji popsán formalismus *DEVS*, jehož je *SmallDEVS* implementací a způsob realizace subsumpční architektury v tomto formalismu. Dále je představen systém pro modelování *BDI* agentů *PNagent* a jazyk *PNtalk*, ve kterém je implementován. V poslední části této kapitoly je zmíněna možnosti napojit *SmallDEVS* na server pro ovládání robotů *Player*, díky jehož architektuře je snadné u těchto systémů realizovat kontinuitu modelu.

Čtvrtá kapitola se již věnuje vybrané úloze nazývané *formování týmu*, jejímu popisu, podrobnostem řešení a implementace v obou srovnávaných variantách – v subsumpční architektuře a v systému *PNagent*.

Kapitola pátá srovnává obě řešení prismatem různých kritérií a následně se snaží vyvodit obecnější závěry k použití daných architektur k inkrementálnímu vývoji agentů s použitím simulace.

V závěru jsou shrnuty dosažené výsledky a z nich vyvozené úvahy na téma vývoje agentů na daných platformách. Jsou zmíněny i možnosti dalšího vývoje implementovaných řešení úlohy a několik návrhů na zlepšení podpory kombinace inkrementálního vývoje a simulace v systému SmallDEVS a PNtalk, nejenom s ohledem na agentní systémy.

[2] Anglicky *model continuity*.

2 Architektury agentů

Pro naše účely přijmeme tuto, v oboru agentních a multiagentních systémů zavedenou, definici agenta (1) :

> *Agent je počítačový systém, situovaný v nějakém prostředí a schopný samostatně jednat v tomto prostředí tak, aby splnil svůj úkol.*

Dále tedy budeme uvažovat architektury – přístupy k návrhu – agentů obecně, bez ohledu na to, zda prostředí, ve kterém se agent pohybuje, je virtuální nebo fyzické. Robot se tedy ve smyslu definice uvedené výše, dá považovat za agenta, situovaného v reálném světě.

Rozdíl mezi virtuálním a reálným prostředím je velice zásadní, neboť komplexnost fyzického světa, ze své povahy vysoce dynamického a nedeterministického, kvalitativně převyšuje jakékoliv člověkem vytvořené virtuální prostředí.

Za inteligentního agenta budeme pokládat takového, jehož chování bychom, jako pozorovatelé, v nějakém smyslu označili za inteligentní. Na konkrétní definici inteligence totiž nepanuje ve vědecké komunitě shoda, nicméně za projev inteligence jsou považovány schopnosti jako např. abstraktní myšlení, uvažování, učení se, plánování nebo užívání jazyka. Pánové Wooldridge a Jennings, kteří se agenty dlouhodobě zabývají, nicméně identifikují následující vlastnosti jako klíčové k označení agenta jako inteligentního (2) :

- *Autonomnost* – schopnost jednat nezávisle, tedy mít pod kontrolou své rozhodování a akce.
- *Reaktivitu* – schopnost vnímat a reagovat na změny v prostředí.
- *Proaktivitu* – schopnost přebírat iniciativu při plnění svých úkolů.
- *Sociálnost* – schopnost interagovat s ostatními agenty.

Architekturu agenta lze definovat jako přístup k návrhu jeho vnitřní struktury. Jsou to tedy principy, kterými je určena dekompozice agenta na komponenty a způsob jejich interakce. Tím je definován mechanismus výběru akce agenta na základě senzorických vjemů a jeho vnitřního stavu, pokud jím disponuje.

V podstatě architektury agentů dělíme na dva proudy podle toho, zda používají model světa, ve kterém se pohybují (jeho reprezentaci) či nikoliv. Agenti, kteří takový model nemají – pracují tedy přímo s okolím – se nazývají reaktivní. Deliberativní jsou potom ti, kteří nějakou formu reprezentace světa používají. Nad tímto modelem provádějí usuzování, jehož výsledek následně promítnou do samotného okolí. Oba přístupy, jak se ukáže, mají svá pro i proti, takže logicky vznikly tzv. hybridní architektury, kombinující oba ideologicky čisté přístupy do inženýrsky pragmatické střední cesty.

2.1 Deliberativní agenti

V základech tohoto přístupu leží představa, že kognitivní funkce u člověka jsou, nebo je alespoň lze modelovat, jako syntaktické manipulace se symboly pomocí transformačních funkcí (odvozovacích pravidel). Tato teze je shrnuta v hypotéze *fyzického symbolického systému* (3):

> *Fyzický symbolický systém má nutné a dostačující prostředky pro obecnou inteligentní akci.*

Deliberativní agenty lze rozdělit právě podle způsobu využití symbolických struktur. Klasickým přístupem je teoretické usuzování, využívající postupů formálních systémů. Modernějším potom využití praktického uvažování, tedy uvažování směřujícího k akci místo k představám.

V cestě k fungujícímu a použitelnému robotovi – agentovi v reálném prostředí – postavenému na symbolické reprezentaci světa stojí dvě překážky (1). Zaprvé – abychom vůbec měli aktuální reprezentaci světa, je nutné zajistit přesný a adekvátní překlad pozorování okolí do symbolické reprezentace, a to dostatečně rychle, aby takové pozorování bylo užitečné. Toto ani po letech intenzivního výzkumu není uspokojivě vyřešeno. Týká se to i jednorázových vstupů (např. jeden obraz z kamery – jeho analýza), ale problém je ještě složitější u dynamických vlastností reálného světa, týkající se především zachycení povahy procesů v okolí probíhajících.

Druhá překážka s první úzce souvisí. Otázkou zde je, jak symbolicky reprezentovat složité entity a jevy okolního světa a jaké postupy použít k uvažování nad těmito reprezentacemi, aby výsledky těchto procesů byly zároveň užitečné a při ukončení těchto procesů ještě aktuální.

2.1.1 Agenti s teoretickým uvažováním

Tato třída agentů používá k symbolické reprezentaci světa logické formule (většinou formule predikátové logiky prvního řádu) a jako nástroj uvažování slouží logická dedukce či dokazování teorémů. Formule, kterými agent disponuje, tvoří bázi znalostí, kterou také můžeme nazvat vnitřním stavem. U báze znalostí není nikdy zaručeno, stejně jako u člověka, že koresponduje se skutečným stavem světa – jsou to tedy představy o světě.

Pokud uvažujeme množinu výroků L predikátové logiky prvního řádu, potom pro bázi znalostí Δ platí, že $\Delta \in \wp(L)$[3]. Usuzování, tedy odvozování nových znalostí, využívá odvozovacích pravidel ρ. Že formule φ může být z báze znalostí odvozena (tedy dokázána) pomocí pravidel ρ, zapíšeme $\Delta \vdash_\rho \varphi$.

Architekturu takového agenta lze abstraktně formulovat pomocí tří funkcí (1):

- $see: S \rightarrow Per$
  Funkce převádějící stav okolí vnímaný senzory (u člověka smyslovými orgány) na jejich vnitřní, symbolickou, reprezentaci – *percepty*. Funkce by měla fungovat rychle a hlavně výstižně, což je u složitějších úloh stále značný problém.

[3] $\wp(L)$ značí potenční množinu – tedy množinu všech podmnožin množiny L

- $next: \wp(L) \times Per \rightarrow \wp(L)$
  Funkce měnící stav báze znalostí na základě současného stavu a zjištěného stavu okolí.
- $action: \wp(L) \rightarrow Ac$
  Funkce výběru akce na základě aktuálního stavu.

Právě funkci výběru akce lze definovat s využitím odvozovacích pravidel. A to tak, že nejlepší možná akce α v daném stavu je taková, pro kterou je možné z báze znalostí odvodit pomocí pravidel ρ formuli $Do(\alpha)$, značící změny způsobené provedením akce. Pokud neexistuje žádná taková akce, pokusíme se najít konzistentní (nezakázanou) akci. Pro tu musí platit, že $\Delta \nvdash_{\rho} \sim Do(\alpha)$. V případě, že nenajdeme ani takovou akci, nezbývá agentovi nic jiného, než zůstat nečinný.

Úkolem návrháře je potom zkonstruovat množinu pravidel ρ, které určují chování agenta. Každé pravidlo bude mít tvar $\varphi(\ldots) \rightarrow \psi(\ldots)$, kde φ a ψ jsou predikáty nad konstantami a proměnnými. Konkrétní takové pravidlo může vypadat následovně:

$$In(x, y) \wedge Dirt(x, y) \rightarrow Do(suck) \qquad \textbf{(2.1)}$$

Chování je tedy popsáno logickou teorií a konkrétní akce se potom vybírají pomocí důkazu v této teorii. Z teoretického hlediska je tento fakt velice zajímavý, protože pokud navrhneme nějakou teorii fungování agenta a specifikujeme ji teorií logickou, jsme schopni tuto specifikaci přímo spustit – bez jakýchkoliv dalších úprav. Tato se potom nazývá *spustitelná specifikace*.

O agentovi využívajícím teoretické usuzování můžeme říci, že je *kalkulativně racionální* – pokud by o svých akcích rozhodoval nekonečně rychle, byl by *perfektně racionální*, tedy volil by vždy nejlepší akci na základě dostupných informací (5).

Výhoda čistě definované sémantiky, dané použitím logik a důkazů v nich, je těžce vyvážena časovou složitostí rozhodovacích procedur dokazatelnosti v těchto logikách. Pro výrokovou logiku je taková rozhodovací procedura v třídě *co-NP*. Tedy abychom zamítli dokazatelnost (tautologičnost) formule, musíme najít jeden negativní příklad, přičemž ověření každého ohodnocení jsme schopni provést v polynomiálním čase (4). Situace u predikátové logiky je ještě nepříjemnější, neboť dokazatelnost je zde obecně nerozhodnutelný problém.

2.1.2 Agenti s praktickým uvažováním

Teoretické usuzování je užití rozumu k rozhodnutí, čemu věřit, či co je pravdivé, k hledání vysvětlení, či k předpovědi věcí budoucích. Je tedy směřované k představám. Agenti s tímto typem usuzováním na základě odvozených dokazatelných představ prováděli akci. Praktické usuzování, na druhou stranu, je užití rozumu směřované k akci. Michael Bratman, na jehož práci o lidském racionálním chování z velké části stojí teorie architektury *BDI*, definuje praktické usuzování následovně (6) :

> *Praktické usuzování je záležitostí zvážení konfliktních úvah pro a proti soupeřícím možnostem, kde relevantní úvahy jsou poskytnuty tím, po čem agent touží, čeho si cení, o co dbá a čemu věří.*

Systémy, jejichž chování lze predikovat právě na základě přisuzování představ, tužeb, záměrů, víry či dalších obdobných pojmů se dle filosofa Daniela Dennetta nazývají *intenční systémy* (7). Tyto lze dělit podle řádu mentálních stavů, kterými systém může disponovat. Systémy prvního řádu jsou takové, kde systém sice má představy a touhy, ale není si jich vědom. Intenční systém druhého řádu již ale má představy a touhy o představách a touhách svých i ostatních agentů. Analogicky lze potom postupovat k systémům vyšších řádů.

Praktické usuzování sestává ze dvou fází. V první – *rozhodovací fázi (deliberation*), jde o rozhodnutí, jakého stavu věcí chceme dosáhnout. V druhé se potom hledá, jak tohoto stavu dosáhnout (v anglické literatuře se označuje termínem *Means-ends reasoning)*.

Na tomto konceptu je přímo postavena zmíněná BDI architektura. Ta modeluje podklady pro agentovo rozhodování pomocí mentálních stavů. Konkrétně *představ* (*belief*), *tužeb* (*desire*) a *záměrů* (*intention*). Architektuře BDI je věnována následující část.

2.1.3 BDI architektura

Abychom mohli proces praktického uvažování (kap. 2.1.2) realizovat, je nutné mít k dispozici informace o tom, co agent chce, čemu věří, apod. – ve smyslu definice praktického uvažování uvedené výše. K modelování tohoto se používají mentální stavy z tzv. *naivní psychologie*[4] – *představa* (*belief*) a *touha* (*desire*). Společně bývají někdy označovány jako *postoje* (*attitudes*).

Představa plní opět roli znalosti, která nutně nemusí být objektivně pravdivá. Jsou to agentovi subjektivní informace o okolním světě. *Touhy* reprezentují motivaci jednání agenta, tedy fakticky jeho cíle. Nepožadujeme u nich ale konzistenci, agentovi touhy tak mohou být protichůdné.

Agenti, kteří se nacházející v dynamických, měnících se prostředích, jsou omezeni prostředky dostupnými k usuzování – především časem. Jedním z předpokladů smysluplného chování agenta v prostředí je tedy efektivní využití dostupných prostředků. Z toho částečně vyplývá, že není možné ani smysluplné, aby se agent rozhodoval o tom, čeho chce dosáhnout, nekonečně dlouho. V určitém momentu se musí rozhodnout pro cíl, který se pokusí splnit. Tento závazek se nazývá *záměr* (*intention*). Záměry mají v praktickém usuzování následující úlohy (1):

- Pohánějí hledání prostředků a způsobů ke svému dosažení. Tedy pokud jsem přijal závazek, pokusím se ho také splnit.
- Záměry jsou persistentní – přetrvávají, dokud nebyly splněny, nebo agent nezačne věřit, že je nelze splnit, anebo agent neuvěří, že důvod pro takový záměr pominul.
- Omezují přijímání dalších závazků – nelze přijmout závazek, který je nekonzistentní s těmi, již přijatými.
- Ovlivňují představy pro budoucí usuzování, protože závazek je přijat s tím, že ho lze dosáhnout a dále tedy agent bude s tímto předpokladem pracovat.

[4] Anglicky *Folk psychology*. Teorie vysvětlující, interpretující a predikující chování na základě běžných mentálních konceptů jako strach, víra, touha nebo představa.

Záměry ve větším měřítku jsou potom *plány* (8). Hledání plánu je vlastně hledání způsobu k dosažení nějakého stavu věcí s využitím dostupných prostředků – tedy druhá fáze praktického usuzování. Plány dále umožňují lepší využití zdrojů, protože zmenšují spektrum možností pro budoucí rozhodování, ale hlavně dovolí koordinovat více komplexních cílů – jak v rámci jednoho agenta (tedy intrapersonálně), tak mezi agenty – interpersonálně. Těžko lze většinou sestavit definitivní dlouhodobý plán, pokud agent operuje v dynamickém prostředí. Plány proto mohou být neúplné (částečné), tvoří potom jakousi kostru, do které jsou hierarchicky doplňovány detailnější plány či konkrétní záměry.

Ačkoliv je praktické usuzování díky tomuto daleko vhodnější pro práci v dynamickém prostředí než teoretické usuzování, stále zůstávají jak problém převodu signálů ze senzorů na vnitřní reprezentaci, aby byla ještě užitečná, tak způsob reprezentace složitých entit a procesů, aby nad nimi bylo možno efektivně provádět manipulace.

2.1.3.1 Formální specifikace

Představy (*beliefs*), *touhy* (*desires*) a *záměry* (*intentions*) je nutné nějakým způsobem modelovat, aby nad nimi bylo možné provádět syntaktické manipulace a usuzování. Tedy dát popsaným filosofickým pojmům konkrétní podobu.

Nejznámějším formálním logickým systémem pro popis mentálních stavů je logika *BDI CTL*. Jejím základem je temporální logika s větveným časem *CTL* (*Computation Tree Logic*). Ta je rozšířena o modální operátory *Bel*, *Des* a *Int*, které jsou definovány nad dvojicí agent – formule. Např. formuli $Bel(a\ \varphi)$ lze neformálně charakterizovat jako část představy o světě agenta a obsahující formuli φ, tedy že agent a věří v platnost této formule (9). Sémantika je, stejně jako v případě samotné CTL, definována s pomocí *Kripkeho struktury*.

```
initialize-state();
while true
   events = percept();
   bel = belrev(bel, events);
   goal = options(bel, int);
   int = filter(bel, goal, int);
   pl = plan(bel, int);
   execute(pl);
   int = dropSuccessful();
   int = dropImpossibe();
end
```

Algoritmus 1 Interpret BDI agenta.

Další formulovanou logikou je *LORA* (*Logic for Rational Agents*), která kombinuje predikátovou logiku prvního řádu, modality pro vyjádření představ, tužeb a záměrů, temporální komponentu pro vyjádření dynamiky systému a akční komponentu pro vyjádření akcí agentů a efektů těchto akcí (10).

Byla také navržena abstraktní architektura BDI agenta, která se snaží přiblížit tento koncept blíže praktické realizaci. Není totiž možné jít cestou přímé realizace formální specifikace, protože dokazovací procedura nad *BDI CTL* i *LORA* logikou je z hlediska času potenciálně neomezená a potlačila by tedy reaktivitu agenta, která je zásadní pro jeho přežití (11). Jádrem architektury je smyčka interpretace agenta, jejíž možná podoba (9) je zapsána v algoritmu 1.

Architektura sestává ze tří dynamických datových struktur reprezentujících představy, touhy a záměry. V každém cyklu se vždy přečtou nové události a na jejich základě se změní představy o světě. Vyberou se dosažitelné cíle a na základě těchto cílů přijmou záměry. Sestaví se plán a tento plán se vykoná. Nakonec se odstraní splněné a nesplnitelné záměry.

2.1.3.2 Procedural Reasoning System

Implementace BDI architektury berou formální základy pouze jako orientační body – právě z důvodů nerealizovatelnosti formálních důkazů v existujících logikách. Příkladem takové implementace je *PRS* (*Procedural Reasoning System*).

K popisu představ tento systém používá predikátovou logiku prvního řádu. Touhy jsou zastoupeny formulemi reprezentujícími chování systému v temporálním smyslu (12). Plány k dosažení jsou zde tzv. *obory znalostí* (*knowledge areas*) uložené v knihovně plánů. Jedná se o deklarativní specifikace procedur, reprezentovatelných jako graf, sestávající z podcílů. V průběhu vykonávání jsou tyto plány částečné i hierarchické – postupně jsou do nich doplňovány plány pro jednotlivé podcíle. *Obory znalostí* mají kromě své struktury ještě spouštěcí podmínku. Ta je využívána pro volbu plánu relevantního ke zvolenému cíli.

Opakující se aktivita agenta potom vypadá následovně (13):

- Je provedeno pozorování světa, které může změnit prostředí a cíle.
- Změna může iniciovat některý plán z knihovny plánů – budou-li splněny podmínky spuštění, tedy plán může vést ke splnění cíle.
- Jeden z těchto plánů může být vybrán a přidán do grafu záměrů (každý záměr tvoří potenciálně do sebe vnořené plány).
- Z grafu je vybrán plán k interpretaci a je z něho vykonán jeden krok.
- Tato akce může ovlivnit prostředí a tedy agentovy představy a cíle nebo změní graf záměrů.

2.1.3.3 AgentSpeak

AgentSpeak je na druhou stranu implementací, která se snaží více přimknout k teoretickým základům. Většina implementací, stejně tak uvedený *Procedural Reasoning System* používala k reprezentaci postojů (představ, tužeb a záměrů) datové struktury místo modálních operátorů (14). AgentSpeak je vlastně jinou formalizací BDI, která může být viděna jako abstrakce minimální verze *PRS*, která dovoluje programovat a interpretovat agenty ve stylu podobném jazyku *Prolog*.

2.2 Reaktivní agenti

Problémy s převodem světa na symbolickou reprezentaci a následným uvažováním nad ní u agentů konceptuálně zakotvených v tradiční umělé inteligenci, vedly k zavržení tohoto směru výzkumu ze strany některých vědců. Ačkoliv výzkum v oblasti umělé inteligence začal s cílem napodobit inteligenci člověka, nakonec se rozpadl na řešení dílčích problémů jako reprezentace znalostí, vidění nebo porozumění přirozenému jazyku. Symbolická reprezentace se stala ústředním tématem, protože zprostředkovávala rozhraní mezi izolovanými subsystémy realizujícími izolované úlohy (15). Argumentuje se také faktem, že symbolické systémy nejsou vůbec podobné tomu, jak fungují biologické systémy – tedy distribuovaně, bez centrálního řídícího prvku, na základě interakce jednodušších komponent.

Alternativní přístupy sdílejí kromě zavržení symbolické reprezentace i další aspekty. Především je to názor, že inteligentní chování nelze vyčlenit jako samostatný předmět zkoumání, ale je výsledkem interakce agenta s prostředím.

Dalším významným znakem je již zmíněný koncept distribuovaného řízení. Inteligentní chování zde není produktem určitého centrálního prvku v kognitivním subsystému, ale povstává[5] z interakce více jednodušších procesů. Podobná tomuto je např. *hypotéza vícera proudů* ve filosofii mysli filosofa Daniela Dennetta týkající se vnímání a vědomí (16) :

> *Různé druhy vnímání, myšlenek a mentálních aktivit jsou v mozku vykonávány paralelními, mnohacestnými procesy interpretace a zpracování senzorických vstupů. Informace vstupující do nervového systému podléhá kontinuálním revizím.*

Jde tedy o snahu stavět agenty přístupem zdola-nahoru, tedy z jednodušších prvků stavět složitější systémy. Těmi jednoduššími prvky jsou zde často *chování*, proto se někdy těmto přístupům říká *behaviorální*. Jindy *reaktivní*, protože agent reaguje přímo na prostředí.

2.2.1 Subsumpční architektura

Subsumpční architektura je bezpochyby nejznámější představitelkou reaktivního přístupu. Dle slov jejího autora Rodney Brookse, jednoho z hlavních oponentů tradičních přístupů, je to paralelní a distribuovaný formalismus pro propojení senzorů a efektorů robota (17). Pracuje bez explicitní reprezentace prostředí i bez explicitního procesu usuzování.

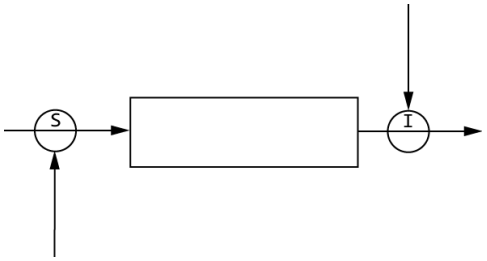
Agent se subsumpční architekturou je dekomponovaný do horizontálních vrstev podle chování. Každá vrstva je tvořena sítí *rozšířených stavových automatů*[6]. Rozšíření konečně-stavového automatu spočívá v přidání časovače, který může být nastaven ke změně stavu po uplynutí daného času. Každý automat má množinu stavů, množinu vstupních a výstupních portů a případně registry pro uchování dodatečných stavových informací. Výstup automatu je funkcí vstupu a hodnot uložených v registrech. Spojení automatů je provedeno propojením výstupních a vstupních portů.

[5] Překlad anglického *emerge*.
[6] Anglicky *Augmented Finite State Machine*.

Návrh řídicího systému v této architektuře je inkrementální proces přidávání nových chování – vrstev. Komplexnost chování agenta se samozřejmě s každou přidanou vrstvou zvyšuje. Sousední vrstvy v modelu interagují pomocí dvou prostředků:

- *Zabránění* (*inhibition*) – zpráva na inhibiční propojce z jiné vrstvy na výstupní port automatu způsobí zabránění výstupu na tomto portu po určitý časový úsek.
- *Potlačení* (*suppression*) – zpráva na potlačovací propojce z jiné vrstvy na libovolný drát je poslána dál po tomto drátu, jako kdyby pocházela od původního zdroje.

Grafickou notaci uvedených propojovacích konceptů vidíte na obrázku 1. Písmenem *I* je značeno zabránění a písmenem *S* potlačení.



Obrázek 1 Grafická notace interakčních mechanismů subsumpční architektury.

Hlavními rysy, kterými se tato architektura vyznačuje a také vymezuje oproti tradičním, symbolickým přístupům, jsou tyto (18):

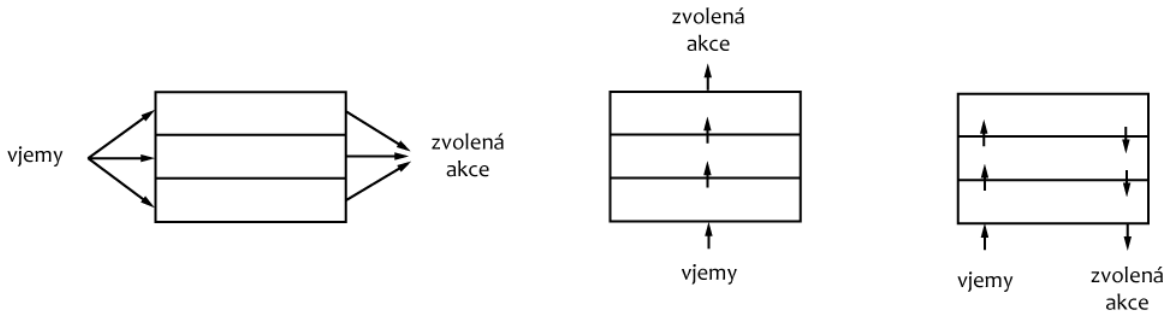
- *Situovanost* – Robot se nachází přímo ve světě. Nezabývá se abstraktními popisy, ale okamžitým vlivem okolí. Svět je svůj nejlepší model.
- *Tělesnost* – Robot má tělo a zkušenost se světem je tedy přímá. Je součástí dynamiky světa.
- *Inteligence* – Inteligence robota je určena dynamikou interakce se světem. Není důsledkem pouze vnitřní struktury agenta, ale také způsobu pozorování a ovlivňování světa a dynamikou světa samotného. Inteligence je vlastnost, kterou přisuzuje vnější pozorovatel.
- *Emergence* – Inteligence vzniká ze složité sítě interakcí zahrnující okolní svět i elementy, z nichž se agenta skládá. Vlivem toho může být obtížné určit kauzální řetězec vedoucí k provedení externí akce agenta.

Ačkoliv tato architektura má zajímavé vlastnosti a při hardwarové implementaci je velice rychlá, její použití má i své nevýhody. Proces návrhu agenta subsumpční architekturou je totiž poměrně zdlouhavý proces, protože interakce vrstev nemusí být vlivem emergence plně předvídatelné. Neznáme tedy přímý inženýrský postup konstrukce složitých robotických systému s touto architekturou. K dosažení požadovaného chování je nutné provést velké množství experimentů a úprav v návrhu.

2.3 Hybridní architektury

Protože oba čisté přístupy – tedy symbolický i reaktivní, mají své nevýhody, bylo poměrně očekávatelné, že se objeví pokusy o propojení obou větví s cílem maximalizovat výhody a minimalizovat omezení.

Hybridní architektury jsou založeny na vrstvení, kde každá vrstva zapouzdřuje nějaké chování či funkcionalitu, zpravidla na rozdílné úrovni abstrakce. Tyto architektury se potom dělí podle způsobu vrstvení a trajektorie toku signálu od senzorů k efektorům (viz. obrázek 2).



Obrázek 2 Typy vrstevnatých architektur. Zleva horizontální, vertikalní jednoprůchodová a dvouprůchodová (1).

Prvním typem je horizontální vrstvení, které známe ze subsumpční architektury. Na rozdíl od ní se zde většinou používá centrální prvek, rozhodující o tom, která vrstva bude nakonec chování řídit. Dalším typem je vertikální vrstvení, které se dále dělí na jednoprůchodové a dvouprůchodové. V jednoprůchodové variantě jsou postupně aktivovány vrstvy od nejnižší k nejvyšší, přičemž tato vygeneruje výstupní akci. V dvouprůchodové nejdříve putuje informace od nejnižší vrstvy k nejvyšší a řízení je potom předáváno směrem dolů. Vyšší vrstvy tak např. mohou delegovat úkoly na ty nižší apod.

Námitkou proti hybridním architekturám, které jsou užitečným a pragmatickým řešením, je že postrádají konceptuální a sémantickou průzračnost, jakou disponují nevrstvená řešení (1).

2.3.1 TouringMachines

Příkladem hybridní architektury s horizontálním vrstvením je architektura *TouringMachines*. Používá tří vrstvy – reaktivní, plánovací a modelovací. Reaktivní implementuje okamžité reakce a je realizována přímým mapováním vstupů na výstupy. Plánovací vrstva rozhoduje o cílech agenta, přičemž k dosažení těchto cílů používá knihovnu parciálních, hierarchických plánů. Modelovací vrstva má potom za úkol, jak název napovídá, modelovat okolí a pomocí něho předvídat konflikty s jinými agenty a generovat na základě tohoto modelu nové cíle pro plánovací vrstvu. Sada těchto vrstev je řízena kontrolním subsystémem, který na základě pravidel může potlačovat senzorické informace pro jednotlivé vrstvy a cenzurovat akce vybrané jednotlivými vrstvami.

2.3.2 InteRRaP

InteRRaP (*Integration of Reactive Bahaviour and Rational Planning*) je vertikální dvouprůchodová architektura. Vrstvy prakticky odpovídají těm v předchozí architektuře – jsou to vrstva chování, plánovací vrstva a kooperační vrstva. Každá vrstva zde má navíc s ní asociovaný model světa na odpovídající úrovni abstrakce. Vrstvy interagují na základě vzájemné aktivace. Nižší vrstva aktivuje bezprostředně vyšší v případě, že není kompetentní k řešení dané situace. Vyšší vrstvy potom mohou využívat možností vrstev nižších k dosahování svých cílů (1).

3 SmallDEVS a vývoj agentů

V této kapitole se budeme zabývat modelovacím a simulačním frameworkem *SmallDEVS* a s na něj napojenými technologiemi a nástroji použitelnými pro vývoj kontrolérů robotů a simulaci robotů jimi řízenými.

Momentálně dostupnými agentními architekturami postavenými či propojitelnými se SmallDEVS jsou systém *PNagent* – implementace BDI architektury v jazyce *PNtalk* – a varianta subsumpční architektury. Společným jmenovatelem obou implementací agentních architektur je formální základ. Ten umožňuje verifikovat vlastnosti agentů, což je poměrně zásadní pro použití autonomních výpočetních systémů v praxi.

3.1 SmallDEVS

SmallDEVS je implementací formalismu *DEVS* (kapitola 3.1.1) postavenou na prototypovém objektovém přístupu, realizovanou v plně dynamickém a objektovém prostředí Squeak Smalltalku. To umožňuje vyšší flexibilitu při modelování, ale především plnou reflexivitu (z hlediska struktury i chování) a otevřenost, kterou plně využijí sebe-modifikující se a adaptivní systémy. Plná otevřenost, tedy dostupnost a manipulovatelnost všech dat v systému, je nutnou podmínkou k adopci interaktivního a inkrementálního vývoje modelů – *průzkumnému modelování* (*exploratory modelling*) (19). Takovýto vývoj modelů v simulaci umožňuje návrháři zkoumat prostor možných modelů a napomáhá tak u komplexních systémů nalézat možná řešení.

SmallDEVS navíc umožňuje heterogenní simulaci - atomické modely zde mohou být popsány místo v DEVS jiným způsobem. Momentálně je možnost popisovat atomické komponenty v jazyce PNtalk, tedy modelovat je pomocí *Objektově Orientovaných Petriho Sítí*.

3.1.1 Discrete Event System Specification

Discrete Event System Specification, zkráceně *DEVS*, je formalismus pro specifikaci matematického objektu zvaného systém (20). Byl vytvořen s cílem umožnit matematicky přesné modelování a simulaci diskrétních systémů, tedy do jisté míry jako protějšek diferenciálních rovnic u spojitých systémů.

Model systému je zde buď atomický, nebo sdružený – někdy nazývaný spojovaný. Spojovaný model odpovídá klasické definici systému, tedy množině entit, které mezi sebou mají vazby – interagují. Entitami jsou zde opět spojované či atomické modely. Systém je tedy modelován hierarchicky.

Atomický model je definován sedmicí $M = (X, S, Y, \delta_{int}, \delta_{ext}, \lambda, \tau)$, kde jednotlivé složky mají následující význam:

- X je množina externích vstupních událostí.
- S je množina sekvenčních stavů.

- Y je množina výstupních událostí.
- $\delta_{int} : S \to S$ je přechodová funkce vnitřního stavu.
- $\delta_{ext} : Q \times X \to S$ je přechodová funkce vnějšího vstupu.
  $Q = \{(s, t_e) | s \in S, t_e \in \langle 0, \tau(s) \rangle\}$ a specifikuje aktuální stav systému a dobu, po kterou se v tomto stavu nachází.
- $\lambda: S \to Y \cup \{\varepsilon\}$ je funkce výstupu, přičemž ε reprezentuje nulový (žádný) výstup.
- $\tau: S \to \mathbb{R}_0^+ \cup \{\infty\}$ je funkce posunu času udávající čas, po kterém je z daného stavu proveden interní přechod.

Spojovaný model, tvořený atomickými i dalšími spojovanými modely, je potom definován n-ticí $N = \big(X, Y, D, \{M_i\}, \{I_i\}, \{Z_{i,j}\}, select\big)$, kde jednotlivé složky jsou postupně:

- X – množina vstupních událostí.
- Y – množina výstupních událostí.
- D – množina jmen komponent.
- Pro každé $i \in D$ je M_i model obsažený v N.
- Pro každé $i \in D$ je I_i množina komponent ovlivňujících M_i.
- $Z_{i,j}$ je funkce zajišťující překlad událostí mezi výstupem M_i a vstupem M_j.Tedy např. pro $i \neq N$ a $j \neq N$ je funkce definována $Z_{i,j} : Y_i \to X_j$.
- $select$ je funkcí výběru komponenty, kterou použije simulátor v případě, že více komponent má naplánovanou událost na stejný čas. V anglické terminologii se označuje jako *tie-breaking function*.

Formálně je pro DEVS definován i algoritmus simulace. Jeho popis lze nalézt např. v (20). Existují i rozšíření formalismu pro paralelní a distribuovanou simulaci, což je užitečné i v případě multiagentních systémů.

3.2 PNagent

PNagent je framework pro modelování agentů s *BDI* architekturou pomocí *Objektově Orientovaných Petriho Sítí*, potažmo jazyka *PNtalk*. Jedná se o otevřený systém, kde stejnými prostředky, jakými je modelován agent – tedy jazykem PNtalk, je tvořen i samotný systém PNagent. Pro vývojáře je tak daleko snadnější přizpůsobit jeho chování konkrétní aplikaci (21).

Systém PNtalk disponuje vizuálními nástroji pro návrh Petriho sítí, a protože se jimi modelují základní prvky architektury určující chování – představy a plány, práce s ním by měla být přehledná a konzistentní. Záměry jsou potom tvořeny instancemi plánů strukturovanými v zásobníku.

Změny v systému jsou prováděny v reakci na události – ty externí reprezentují změny v prostředí a interní jsou důsledkem dynamiky vnitřních procesů agenta. Cíle (touhy) jsou zde také modelovány jako události, které jsou ale perzistentní. Místo jednorázového zpracování se k nim systém vrací, dokud nebyly splněny nebo se nestaly neplatnými.

Klasický cyklus interpretace BDI agenta je zde rozdělen do dvou částí – jedna je zodpovědná za zpracování událostí – tedy výběr plánů, které jsou touto událostí spustitelné a také správu existujících plánů. Ve druhé části potom probíhá samotná interpretace struktury záměrů. Volné prokládání obou částí je možné díky vlastnostem Petriho sítí (22).

3.2.1 PNtalk

PNTalk je jazyk vycházející z Petriho Sítí, konkrétně *Barvených Petriho Sítí*[7]. Petriho Sítě jsou formalismem pro popis distribuovaných diskrétních systémů. Jejich výhodou oproti ostatním formalismům je ekvivalentní grafické reprezentace v podobě bipartitního grafu.

Výchozím bodem je P/T síť definovaná šesticí $N = (P, T, F, W, K, M_0)$, kde složky mají následující význam (22) :

- P je konečná množina míst.
- T konečná množina přechodů.
- $F \subseteq (P \times T) \cup (T \times P)$ je množina hran.
- Zobrazení $W: F \to \mathbb{N} \setminus \{0\}$ je ohodnocení hran grafu sítě určující váhu každé hrany.
- Zobrazení $K: P \to \mathbb{N} \cup \{\omega\}$ specifikuje kapacitu míst (v případě ω neomezenou).
- Zobrazení $M_0: P \to \mathbb{N} \cup \{\omega\}$ je počáteční značení míst sítě, přičemž platí formule $\forall p \in P: M_0(p) \leq K(p)$.

Stav sítě je definován distribucí značek v místech. Přechod se může spustit, pokud ve všech místech připojených hranou jako vstup, je dostatečný počet značek definovaných váhovou funkcí a v cílových místech je dostatečná kapacita. Formálně zapsáno je přechod t proveditelný ve značení M, pokud platí $\forall p \in {}^{\cdot}t: M(p) \geq W(p,t) \wedge \forall p \in t^{\cdot}: M(p) \leq K(p) - W(t,p)$. V takovém případě se značky odeberou a počet značek (opět daný váhovou funkcí) je umístěn na každé místo připojené hranou jako výstupní.

Rozšířením P/T sítí jsou *Barvené Petriho Sítě* umožňující přiřadit značkám typ (barvu) a pomocí inskripčního jazyka s nimi na hranách či v přechodech manipulovat. Přechody je navíc možno opatřit strážemi, které dále omezují podmínky jejich spuštění.

PNtalk je potom realizací *Objektově Orientovaných Petriho Sítí* – hierarchických sítí vycházejících z Barvených Petriho Sítí. Lze s nimi modelovat objekty konceptuálně odpovídající objektovému systému v prostředí *Smalltalk* (23), jehož jazyk také používají jako inskripční. Jedná se tedy o objekty komunikující pomocí zpráv, reagující s dynamickou vazbou a instanciované z tříd organizovaných v hierarchii dědičnosti. Třída popsaná jazykem PNtalk sestává ze sítě objektu, definující reprezentaci instancí třídy a jejich nezávislou aktivitu, a množiny sítí metod definujících reakce na příchozí zprávy (23). Instance takovéto třídy je potom definována instancí objektové třídy a jejím stavem. Pro každou příchozí zprávu je dynamicky instanciována obslužná síť, přičemž je samozřejmě užit koncept polymorfismu.

[7] Coloured Petri Nets (CPN)

Objekty modelované PNtalkem je možno umisťovat do DEVS modelů, díky možnosti obalit je wrapperem[8], který má vnější rozhraní kompatibilní se SmallDEVS modely a překládá příchozí zprávy tak, aby je vnitřní objekt dokázal zpracovat. Je tak možno objekty v jazyce PNtalk snadno začlenit do simulací a tím ve SmallDEVS realizovat multiparadigmatickou simulaci.

3.3 Realizace subsumpční architektury

Nad definicí atomického *DEVS* modelu lze snadno postavit model rozšířeného konečného automatu, tedy základního stavebního prvku subsumpční architektury (viz. kapitola 2.2.1). Přechodová funkce vnějšího vstupu je vlastně přechodovou funkcí konečného automatu – přesně dle definice, pokud v ní zanedbáme čas uplynulý od přechodu do aktuálního stavu. Přechodovou funkce vnitřního stavu lze potom využít k realizaci časovače měnícího stav po uplynutí určitého časového úseku.

Koncepty zabránění a potlačení signálů je možno implementovat jako atomické modely. Nicméně prozatím bylo možné implementovat agenty bez jejich použití nebo tyto mechanismy implantovat do komponent realizujících chování.

3.4 SmallDEVS a simulace agentů

Nutnou podmínkou použitelnosti SmallDEVS a s ním spjatých technologií k tvorbě kontrolérů fyzických agentů (robotů) v simulaci, je možnost napojit je na simulátor prostředí. Ve SmallDEVS byl vytvořen klient k serveru pro ovládání robotů *Player*[9]. Ten běží na skutečném robotovi a umožňuje vzdáleně, přes síťové rozhraní, sledovat a ovládat jeho senzory resp. efektory. Klient, nazývaný také rozhraní, je realizován jako spojovaný DEVS model, který zajišťuje síťovou komunikaci a překlad mezi vybranými konstrukcemi v jazyce Smalltalk a komunikačním protokolem Playeru.

Velkou výhodou je možnost napojit Player, místo na skutečného robota, na model v simulovaném prostředí. K dispozici jsou dva simulátory lišící se svým zaměřením a tudíž přístupem k problému. Prvním je *Stage*, který budeme používat, a který nabízí jednoduchý dvourozměrný svět, umožňující výpočetně relativně nenáročnou simulaci i velkého počtu modelů. Nesnaží se přitom dosáhnout vysoké věrnosti prostředí vzhledem k realitě, tedy skutečného chování senzorů a efektorů. Druhým simulátorem je *Gazebo*, který nabízí trojrozměrný svět s realistickou odezvou senzorů a na fyzikálních modelech postavené interakce mezi objekty.

Toto řešení by mělo být dostatečné k interaktivnímu vývoji agentních a multiagentních systémů v simulaci. Přičemž možnost napojit server Player jak na skutečného robota, tak na jeho simulovanou verzi, nám otvírá dveře pro kontinuitu modelu – tedy pro použití stejného modelu při vývoji i při skutečném provozu.

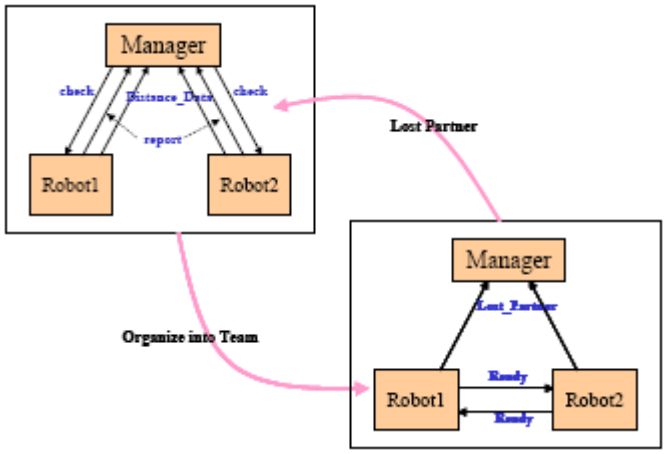
[8] Odvozeno od návrhového vzoru Wrapper, jehož je popisovaná konstrukce konkrétní realizací.
[9] http://playerstage.sourceforge.net/

4 Úloha formování týmu

Jako úloha, na níž bude provedeno srovnání, byla vybrána robotická úloha nazývaná *Formování týmu* nebo také *Vůdce-Následovník* (v originále *Team formation* resp. *Leader-Follower*). Tato je převzata z článku *Model Continuity to Support Software Development for Distributed Robotic Systems: A Team Formation Example* (24) autorů Bernarda P. Zeiglera a Xiaolin Hu.

Úloha popisuje distribuovaný robotický systém – bez centrálního řízení, v němž v základní verzi figurují dva roboti (lze ji samozřejmě zobecnit a formulovat ji tak pro obecný počet robotů). Výchozím stavem jsou roboti pohybující se autonomně v nějakém prostředí, přičemž vzájemně vědí o svojí existenci a snaží se jeden druhého najít pomocí informací ze svých senzorů. V momentě, kdy se rozpoznají, zkoordinují vzájemnou polohu a jeden z nich se stane vůdcem a druhý následovníkem přesně kopírujícím vůdcovi pohyby. Tímto koordinovaným způsobem se pohybují po prostředí, dokud z nějakého důvodu neztratí vzájemný kontakt. V tom případě se opět stávají autonomními a vracejí se tak do fáze hledání.

V původním článku v modelu systému kromě robotů figuruje ještě softwarová komponenta zvaná Manager (viz obrázek 3). Ta má za úkol porovnávat informace ze senzorů jednotlivých robotů, koordinovat proces vzájemného rozpoznání a iniciovat strukturální změny v modelu vedoucí k utvoření či rozpuštění týmu.



Obrázek 3 Model systému "Leader-Follower" z článku (23).

4.1 Motivace výběru

Formování týmů splňuje základní požadavek – tedy je robotickou úlohou. Navíc, kromě samotného řízení robota, obsahuje i úroveň inter-robotické interakce a komunikace. Tato částečně posouvá systém do oblasti obecných multiagentních systémů, což může být pro zamýšlené srovnání užitečné.

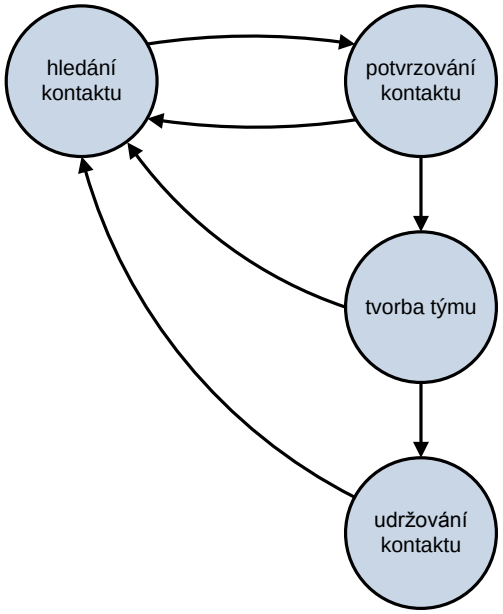
Hlavním faktorem ovšem byla její distribuovatelnost. Roboti musí fungovat jako autonomní entity a mohou tedy být řízeni z různých, na sobě nezávislých systémů, přičemž spolu musí aktivně

komunikovat. Proto je oprávněné předpokládat, že model řízení dosáhne určité úrovně komplexnosti, pří které se projeví charakteristické vlastnosti agentních architektur a systémů, ve kterých jsou tyto architektury realizovány. Díky této vlastnosti bude implementace této úlohy moci být využívána k testování distribuované simulace v dalších iteracích vývoje systému SmallDEVS.

Lze si i reálně představit další rozšiřování úlohy co do počtu robotů vyskytujících se v prostředí. Tím je možno dále zvyšovat komplexnost modelu a především celkovou komunikační intenzitu v systému. Tím se otevírají dveře dalším experimentům a testům.

4.2 Analýza, předpoklady a východiska řešení

Jak bylo řečeno, úloha *Vůdce-Následovník* popisuje distribuovaný robotický systém (25) – tedy takový, jenž neobsahuje centrální prvek, který by ho jako celek řídil. Celkové chování je definováno jako kompozice chování jednotlivých autonomních prvků v systému. V tomto případě dvou autonomních robotů. Toto východisko je ale, dle mého názoru, v řešení v původním článku porušeno – softwarový *Manager* je komponentou, která fakticky koordinuje úkony v procesu sestavování a rozpouštění týmu. Roboti jsou potom paradoxně nejvíce samostatní, co se propojení týče, až v momentě, kdy zformovali tým – tedy jeden pouze kopíruje pohyby druhého.



Obrázek 4 Globální stavový prostor systému úlohy Formování týmu.

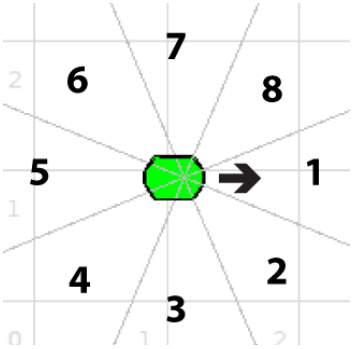
Vzájemné rozpoznání a iniciace strukturálních změn definujících novou roli robotů musí být čistě v rukou robotů samotných – bez prostředníka. Jediným možným prostředkem splňujícím tyto požadavky je přímý komunikační kanál mezi agenty, pomocí něhož mohou vzájemné akce koordinovat. Komunikační kanál umožňuje snadnou distribuovatelnost systému – roboti mohou běžet v oddělených simulacích. Může být v budoucnu realizován např. i přenosem informací prostředím, ve kterém se roboti pohybují (např. pomocí zvuku, konkrétně reproduktoru a

mikrofonu). Tím by byly odstraněny všechny virtuální elementy a propojení a roboti by tím byli skutečnými autonomními entitami.

U tohoto systému lze identifikovat čtyři globální fáze, kterými systém prochází. Na počátku jsou roboti autonomní a vzájemně se hledají a nějakou formou spolu komunikují. V případě, že se jeden z nich domnívá, že má s druhým kontakt, pokusí se koordinovaně tento kontakt potvrdit. V případě, že se potvrdí, přechází systém do další fáze, ve které se snaží utvořit formaci vhodnou pro společný pochod. Poslední fází je právě společný pochod, který se rozpadá, pokud spolu roboti ztratí kontakt. To je zpravidla způsobeno tím, že následovník není schopen vykonat nařízený pohyb. V tom případě se systém vrací do fáze vzájemného hledání. Stejně tak musí existovat přechod z potvrzování kontaktu do jeho hledání, pokud se kontakt potvrdit nepodaří. Možný návrat z fáze tvorby týmu do fáze hledání může obdařit systém vyšší robustností. V podobě stavového diagramu je globální stavový prostor systému na obrázku 4.

4.2.1 Senzorické vybavení robota

Pokud roboti nebudou disponovat kamerou nebo jiným mechanismem (např. echolokací[10]), poskytujícím dostatečné množství informací o blízkém okolí, z nichž by bylo možné rozpoznávat objekty v tomto okolí se nacházející, jsme odkázáni pouze na informace ze senzorů měřících vzdálenost překážky v určitém směru – např. sonar či laser. Z nich ovšem nejsme schopni rozpoznat, zda je překážka druhým robotem či nikoliv. Abychom tedy umožnili vzájemné rozpoznání je nutno zajistit, že se v daném prostředí nebudou nacházet jiné pohybující se objekty než samotní fyzičtí agenti.



Obrázek 5 Rozložení sonarů na robotu a jejich indexy v prostředí Smalltalk.

[10] Echolokace je smysl, kterým disponují například netopýři či delfíni. Je založen, podobně jako sonar, na principu vyluzování ultrazvuku a vyhodnocování následných ozvěn. Pokud zvíře disponuje více přijímači, může fakticky „vidět“ prostorově a díky tomu i identifikovat zachycené objekty.

Dalším druhem senzoru, který může figurovat jako vstup pro procesy řídící pohyb robota je nárazník (anglicky bumper[11]). Nárazník reaguje na své stisknutí, je tudíž schopen detekovat, zda robot narazil do překážky. Při experimentování s robotickým simulátorem Stage se ovšem senzor nechoval spolehlivě a často nedetekoval náraz. V návrhu robota je proto jeho úloha oslabena na instanci poslední záchrany.

Robot tedy ve finální konfiguraci disponuje osmi sonary rozloženými rovnoměrně po obvodu robota, které tak zabírají celých 360° (viz. obrázek 5). Po obvodu je také připevněno osm nárazníků, které umožní robotu reagovat na nárazy do překážky. Výskyt této eventuality by ale kvůli nespolehlivosti nárazníků měl být minimalizován způsobem řízení pohybu.

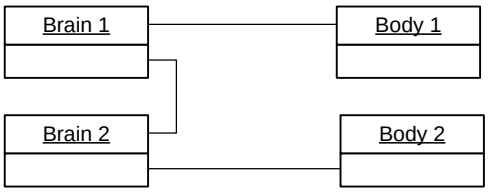
Player nabízí navíc ještě informace o aktuální relativní poloze a rychlosti robota. Zjištění těchto údajů probíhá na principu odometrie – měřením ujeté cesty a kombinací s počáteční polohou. Tato metoda je ale značně náchylná k akumulaci chyby – tedy s ujetou vzdáleností se chyba zvětšuje (26). To může být při potřebě přesné navigace na obtíž.

4.3 Řešení subsumpční architekturou

Subsumpční architektura patří do rodiny reaktivních architektur, přičemž dekomponuje řešení na horizontální vrstvy implementující postupně vyšší úrovně chování s možností se vzájemně ovlivňovat (viz. kapitola 2.2.1). Postupně je tedy nutné implementovat jednotlivá chování směrem ze zdola nahoru, od těch jednodušších, s průběžným ověřováním korektnosti chování celého systému v simulaci.

4.3.1 Konceptuální návrh

I přesto, že proces vývoje bude využívat inkrementálního modelování jako prostředku k hledání možných řešení, je nutno vycházet z určitého konceptuálního návrhu, který je samozřejmě do značné míry ovlivněn vybranou agentní architekturou a řešeným problémem.

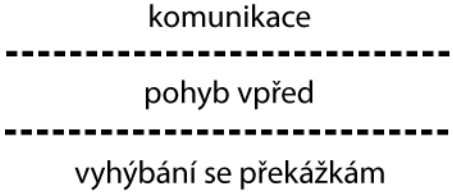


Obrázek 6 Model systému s kontroléry robotů (mozky) a komponentami zapouzdřujícími rozhraní na Player (těly).

[11] Od anglického slova *bump* – rána či od slovesného tvaru *to bump* – narazit.

Nejdříve se podívejme na celkový model systému (obrázek 7). Ten je zamýšlen tak, aby byl přehledný a snadno interpretovatelný. O řídicích systémech robotů je totiž možno přemýšlet jako o mozku (brain), který řídí tělo (body). To se v modelu dá reprezentovat jako rozhraní na robotický server Player. Komunikační kanál bude, alespoň v zamýšlené variantě tvořit propojení mezi mozky robotů.

Nyní se věnujme koncepci kontroléru robota. Protože stavíme na subsumpční architektuře, budeme vycházet z vrstveného modelu. Nejnižší vrstva zajišťuje samotný pohyb robota po prostředí. Tento zahrnuje dvě složky – vyhýbání se překážkám a pohyb ve volném prostoru. Aby bylo učiněno zadost filosofii subsumpční architektury, je vhodné implementovat každou složku jako speciální chování (vrstvu). Pohyb jako takový lze simulovat samostatně s použitím jednoho robota – tedy jako základ, na kterém bude dále stavěno.



Obrázek 7 Konceptuální dekompozice modelu pro subsumpční architekturu.

Nad vrstvami implementujícími pohyb stojí vrstva zajišťující komunikaci robotů, která implementuje komunikační protokol, umožňující vzájemnou výměnu dat ze senzorů a příkazy, pomocí nichž bude možno provést potvrzení vzájemného kontaktu a iniciaci změn rolí agentů.

4.3.2 Chování a komunikační protokol

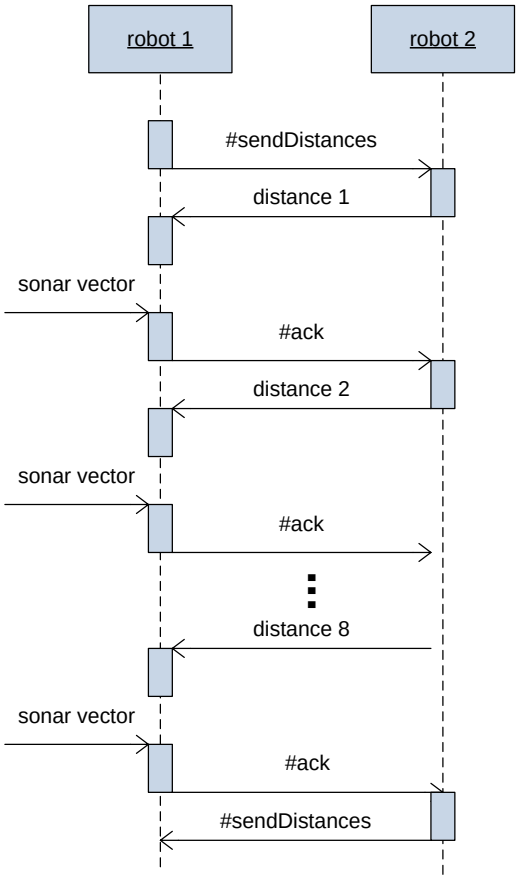
U každého systému, jehož součástí je komunikace a je tedy nutno navrhnout komunikační protokol, musí návrhář zvážit způsob, jakým tato komunikace bude probíhat – zda bude implicitně spoléháno na korektní doručení zpráv a na příslušnou reakci na druhé straně, nebo zda bude žádáno potvrzení. Komunikace bez potvrzování je méně náročná na komunikační kanál, ale je náchylnější k chybám a vyžaduje tedy robustnější řízení a komunikační protokol, který se systému umožní vyrovnat s potenciálními nekonzistentními či nekorektními stavy. Celkově je robustnější řízení samozřejmě výhodnější, nicméně návrh takového řízení je komplikovanější.

Protože již takto bude celkový systém řízení relativně komplexní, byla zvolena varianta komunikace s potvrzováním, která umožní sekvenční a synchronní změny stavů komponent tvořících řídicí systémy robotů.

4.3.2.1 Fáze hledání kontaktu

Výchozím stavem obou robotů je autonomie. Průběžně si roboti vyměňují vzdálenosti naměřené sonary a porovnávají je se svými hodnotami za účelem zjištění, zda se nenacházejí ve vzájemném dosahu. Celý tento proces byl navržen s důrazem na dvě charakteristiky – reaktivnost a symetrii rolí obou robotů.

Reaktivností je myšlena minimalizace časové náročnosti výpočetních procesů, nutných ke zpracování přijaté zprávy s hodnotami sonarů druhého agenta. Pokud by tyto trvaly příliš dlouho, reakce na stav prostředí by nemusely být aktuální či by dokonce nemusely být provedeny vůbec. Měřením byla odhadnuta frekvence příchozích aktualizací hodnot na sonarech od rozhraní na Player na 10 Hz. Na základě experimentů bylo učiněno rozhodnutí, že je to příliš vysoká frekvence na to, aby byla vždy posílána a porovnávána data ze všech sonarů. Aby nebyla ohrožena reaktivnost robota dlouhými výpočty, bude posílána vždy pouze hodnota z jednoho sonaru. Tento přístup omezí nejvyšší počet porovnání v jednom cyklu na $|\overrightarrow{s_m}| = 8$ (velikost vektoru obsahující aktuální hodnoty sonarů, fakticky počet sonarů)[12]. Při zasílání celého vektoru by počet porovnání byl roven $|\overrightarrow{s_m}| \cdot |\overrightarrow{s_o}| = 64$ (počet vlastních sonarů násoben počtem sonarů druhého robota).



Obrázek 8 Sekvenční diagram komunikace při vzájemné výměně vzdáleností. Důležité je, že poslední zprávou se role robotů obracejí.

Navíc i proces potvrzování byl navázán na příchozí data ze sonarů – fakticky je tedy možno říci, že celý proces komunikace je, až na výjimky, taktován signálem ze sonarů. Je tak možno přizpůsobovat reaktivnost robota možnostem výpočetního zařízení, na kterém běží, nastavením frekvence aktualizací ze serveru Player.

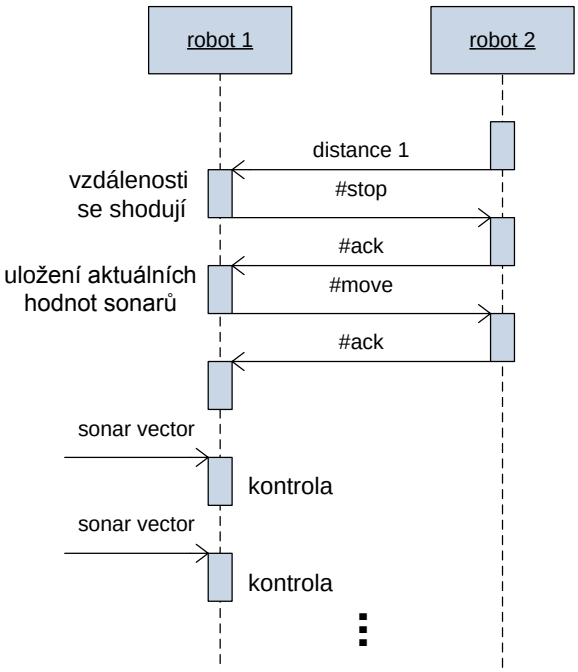
[12] Vzájemná orientace robotů může být libovolná, je tedy nutno kontrolovat všechny možné kombinace sonarů.

Symetrie rolí poukazuje na skutečně rovnocenné postavení obou robotů při dlouhodobém pohledu na proces komunikace. Průběžně se musí střídat role zasílajícího a porovnávajícího. Jak bude popsáno dále, tímto je i vyrovnána šance obou robotů stát se vůdcem týmu. Možný průběh komunikace v průběhu fáze detekce kontaktu je zachycen na sekvenčním diagramu na obrázku 8.

4.3.2.2 Fáze potvrzování kontaktu

V momentě, kdy dojde ke shodě[13] vzdálenosti na cizím sonaru s jedním z vlastních, jsou oba roboti z pokynu porovnávajícího zastaveni (druhému je zaslána zpráva *#stop*). Po potvrzení zastavení jsou uloženy aktuální vzdálenosti na sonarech a druhému robotu je zaslána zpráva *#move*, který se v reakci na ní začne opět pohybovat (viz. obrázek 9). Porovnávající robot potom kontroluje, zda se hodnoty na jeho sonarech nezmění. Protože byl vysloven předpoklad, že v prostředí není kromě robotů žádná pohybující se entita, změna může být způsobena jedině pohybem druhého robota.

Pokud v určené době (dané množstvím přijatých aktualizací sonarů) je detekována změna, kontakt je tím potvrzen a je zahájena fáze tvorby týmu, tedy přípravy ke společnému pohybu. Naopak, pokud se změna polohy druhého robota na měřených vzdálenostech neprojeví, je i porovnávající robot opět uveden do pohybu a stav systému tak opět přechází do fáze výměny dat.



Obrázek 9 Fáze detekce kontaktu a následného potvrzování.

4.3.2.3 Fáze tvorby týmu

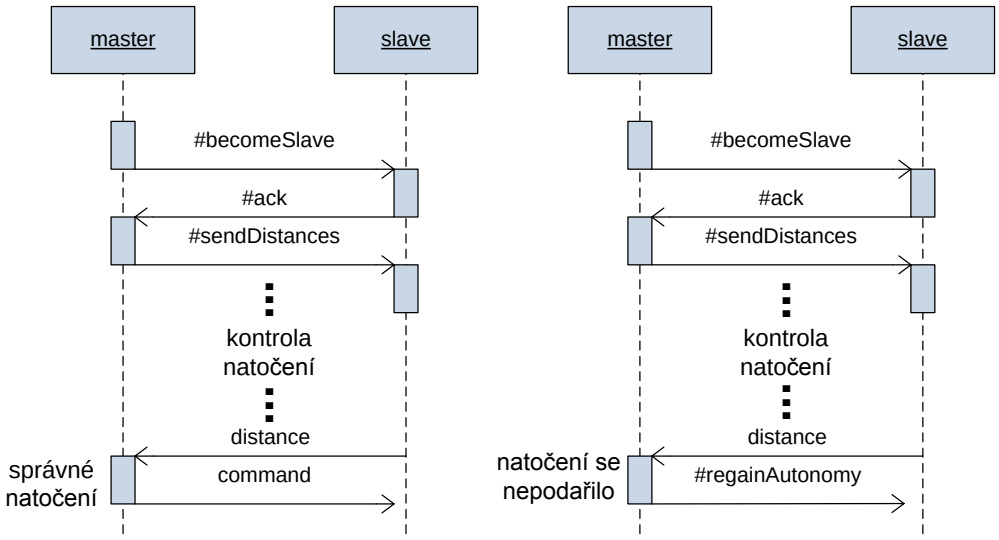
V případě, že byl detekován a potvrzen vzájemný kontakt obou robotů, ten který detekci provedl, se přetransformuje na vůdce (dále tento stav bude označován *Master*[14]) a zasílá druhému zprávu

[13] Samozřejmě s jistou odchylkou, protože roboti se pohybují a ani senzor nemusí být úplně přesný.
[14] Místo *Leader-Follower* bude používáno *Master-Slave* – kvůli snadnějšímu převedení do slovesných tvarů používaných v implementaci (např. enslave).

#enslave. Tato zpráva je de facto příkazem k provedení strukturálních změn vedoucích k přijmutí role následovníka (*Slave*) a k následnému otáčení se na místě. *Master*, po potvrzení zprávy *#enslave*, průběžně kontroluje, zda jsou oba natočeni stejným směrem. Kontrola probíhá hledáním takového natočení, aby stejná vzdálenost byla u jednoho robota na sonaru s indexem *i* a u druhého s indexem *i+4*. Takto indexované sonary jsou sonary protilehlé (viz. kapitola 4.2.1, obrázek 5). Kontrolu navíc provádí jen *Master* – v případě, že obdrží zprávu *#sendDistances*, vrací ji okamžitě zpět. Tento koncept je blíže popsán v části týkající se fáze udržování kontaktu (4.3.2.4).

Celý proces je stejně jako u potvrzování kontaktu časově omezen (opět navázáním na počet aktualizací stavu sonarů). Pokud se v tomto časovém úseku nepovede natočení provést, je *Masterem* zaslána zpráva *#regainAutonomy*, na kterou robot reaguje převedením své struktury zpět do výchozího (autonomního) stavu.

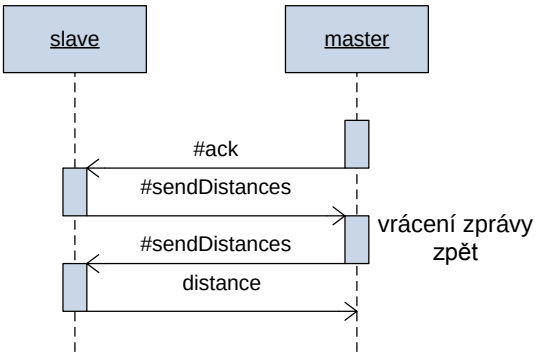
Pokud vzájemné natočení odpovídá danému požadavku, *Master* se uvede do pohybu a díky strukturálním změnám je tento příkaz poslán druhému robotovi – ten tedy vykonává stejný pohyb.



Obrázek 10 Sekvenční diagram fáze hledání stejného natočení. Vlevo zobrazena varianta, kdy je nalezeno. Vpravo potom když není nalezeno v daném časovém úseku.

4.3.2.4 Fáze udržování kontaktu

Ve chvíli, kdy je tým vytvořen a probíhá společný pochod, je nutné průběžně kontrolovat, zda na sebe stále roboti mají sonarový kontakt. Toto opět probíhá pomocí zasílání dat ze sonarů – podobným způsobem jako ve fázi hledání kontaktu. První rozdíl je ovšem v tom, že nyní čekáme, že kontakt detekován bude. K akci tedy dochází, pokud po nějaký časový úsek kontakt není potvrzen. V tom případě je tým rozpuštěn – následovníkovi je zaslána zpráva *#regainAutonomy* a *Master* sám sebe převede zpět do autonomního stavu. Celý systém je tak převeden do fáze hledání kontaktu.



Obrázek 11 Vracení zprávy #sendDistances ve fázi udržování kontaktu.

Druhým rozdílem oproti fázi hledání kontaktu je asymetrie rolí. Nyní nejsou oba roboti autonomní, ale jeden má nad druhým kontrolu. Veškerá zodpovědnost za porovnávání sonarových dat je proto přenesena na toho, který je ve stavu *Master*. Co je u něho upraveno, je reakce na zprávu *#sendDistances* – místo aby začal posílat data ze svých senzorů, vrací okamžitě zprávu zpět (viz. obrázek 11).

4.3.3 Popis výsledného DEVS modelu

V této části budou popsány všechny důležité aspekty výsledného modelu řídicího systému robota – od vrstev zajišťujících pohyb až po komunikační vrstvu. V závěru bude také zmíněna podoba modelu celé úlohy – tedy propojení robotů a jejich napojení na simulované prostředí.

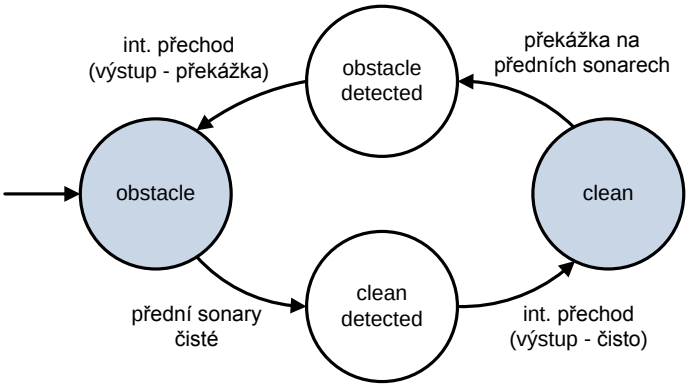
Model byl implementován ve frameworku SmallDEVS, tedy pomocí formalismu DEVS, přičemž bylo využito zjednodušující nadstavby, umožňující atomické komponenty popisovat v intencích konečně-stavových automatů.

4.3.3.1 Vrstvy zajišťující pohyb

Při návrhu vrstev bylo vycházeno z již zmíněné nespolehlivosti bumperů (detektorů nárazu) při použití simulátoru Stage. Jejich role byla tedy oslabena na instanci poslední záchrany při nárazu. Primárním senzorem využívaným pro pohyb se tedy stal sonar. Vzdálenosti překážek před robotem – na čelních sonarech (hodnoty na levém předním, čelním a pravém předním sonaru) jsou určující pro charakter pohybu vykonávaného robotem. Pokud jsou vzdálenosti příliš malé, je aktivováno vyhýbání se s účelem najít volnou cestu. Pokud jsou překážky dostatečně vzdálené je možno postupovat vpřed.

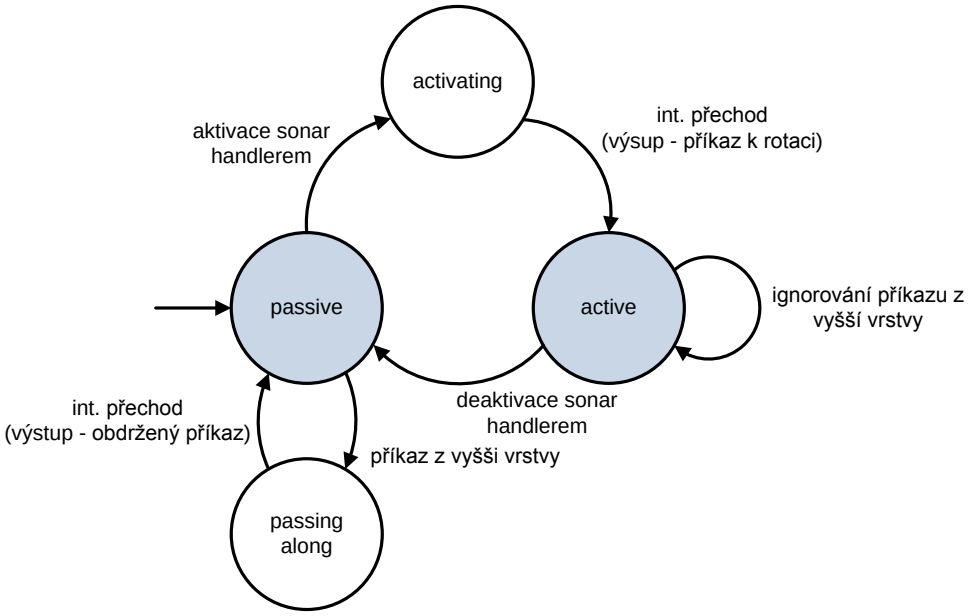
Protože podmínky aktivace vrstev *Vyhýbání se překážkám* (anglicky označovaná jako *Avoid*) a *Pohybu vpřed* (anglicky *Wander*) je možno považovat za komplementární – jedna je aktivována, když druhá ne a obráceně – v průběhu implementace se ukázalo jako výhodné předřadit před komponenty implementující zmíněné vrstvy komponentu, předzpracovávající data ze sonarů a dodávající jim tak jen nezbytně nutné údaje v nezbytně nutných případech. Tato komponenta byla nazvána *Sonar Handler*.

Sonar Handler se může nacházet ve dvou stavech – před robotem se nachází překážka nebo prázdno. V případě, že je před ním překážka, je aktivována komponenta *Avoid*, která začne robotem otáčet. Ve chvíli, kdy jsou na čelních sonarech dostatečně vysoké hodnoty, je aktivována komponenta *Wander* a robot se začne pohybovat vpřed. Stavový diagram komponenty *Sonar Handler* vidíte na obrázku 12.



Obrázek 12 Stavový diagram komponenty Sonar Handler

Komponenta *Wander* je po této redistribuci funkcionality vlastně degradována na bezestavovou komponentu, umožňující pomocí jí zaslaných zpráv robota rozejít směrem vpřed a zastavit ho. Důležitým parametrem pohybu je přitom jeho rychlost. Je nutno si uvědomit, že reakce systému na vstup není okamžitá, ale trvá určitou dobu. Agent by při příliš vysoké rychlosti nemusel stihnout zareagovat a mohlo by dojít ke kolizi či k jiným problémům v případě vyšších vrstev.



Obrázek 13 Stavový diagram komponenty Avoider

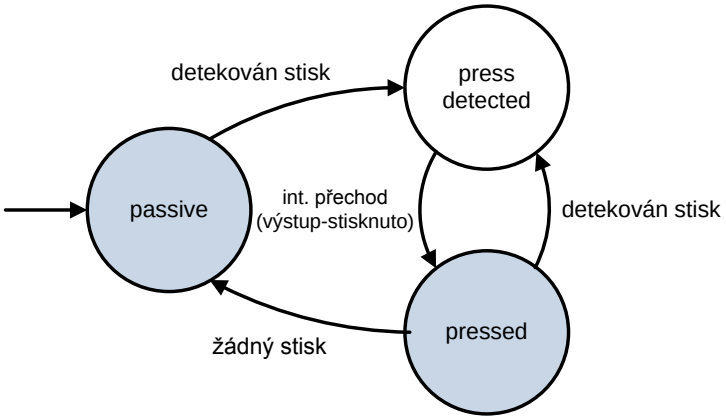
U komponenty *Avoid*, realizující vyhýbání se překážkám, je stavový prostor komplikovanější, což je dáno dvěma faktory. Zaprvé musí být do řízení začleněny i bumpery – jako poslední instance, pokud skutečně dojde k nárazu. Tento fakt vnitřně dělí *Avoid* na dvě části – *Bumper Handler* a *Avoider*.

Zadruhé je to předpoklad, že vyhnutí se kolizím je základním předpokladem dosažení vyšších cílů – tedy nalezení druhého robota a vytvoření týmu. To jasně dává *Avoider*u nejvyšší prioritu při vydávání pokynů k pohybu. Musí tedy být schopen potlačit příkazy (pokusy o převzetí řízení) pocházející z vyšších vrstev v případě, že je momentálně aktivní – pohyb vyvolaný *Avoider*em nesmí být přerušen.

Pro ujasnění to shrňme – pokud komponenta není aktivována, propouští příkazy z vyšších vrstev. Pokud je aktivní – je vykonáván pohyb zabraňující nárazu – příkazy shora jsou ignorovány. V podobě stavového diagramu je komponenta zobrazena na obrázku 13. Je nutné si uvědomit, že *Avoid* je kauzálně závislý na *Sonar Handleru*, který je fakticky aktivátorem a represorem komponenty – hraje tedy primární roli v přechodech mezi jednotlivými stavy.

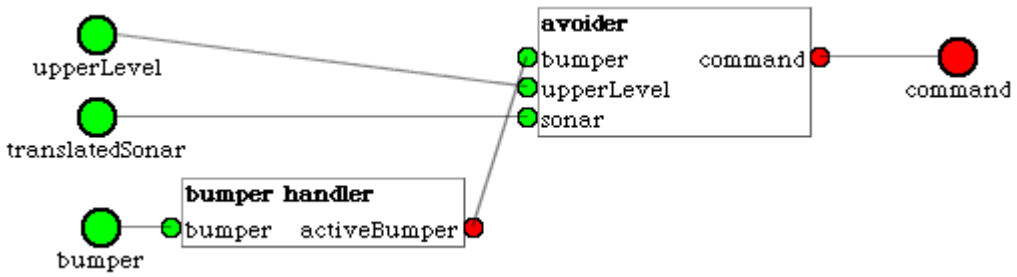
Samotné vyhýbání je implementováno jako otáčení se na místě, a to ve směru levého či pravého předního sonaru, podle toho, na kterém je aktuálně vyšší hodnota – v jehož směru je překážka vzdálenější.

Zbývající komponentou je zmíněný *Bumper Handler* – obsluha dat z bumperů. Její stavový prostor reflektuje povahu samotného senzoru – stisknuto/nestisknuto. Ve verzi před přidáním komunikačních vrstev dokonce existovaly pouze tyto dva stavy, přičemž *Avoider* byl informován jen při změně stavu. Při zavedení stavu systému, kdy jeden robot následuje příkazy druhého bez ohledu na vlastní senzory, ovšem může dojít k nárazu „otroka“ do překážky. V momentě, kdy dojde k rozpuštění týmu, nemá ale *Avoider* na co reagovat – informace o detekci kolize již byla vyslána, ale robot sám sebe neřídil. Proto byla provedena úprava, aby informace o detekci byla posílána opakovaně a robot se tak s touto situací mohl vyrovnat. Výsledný stavový diagram komponenty je vyobrazen na obr. 14.



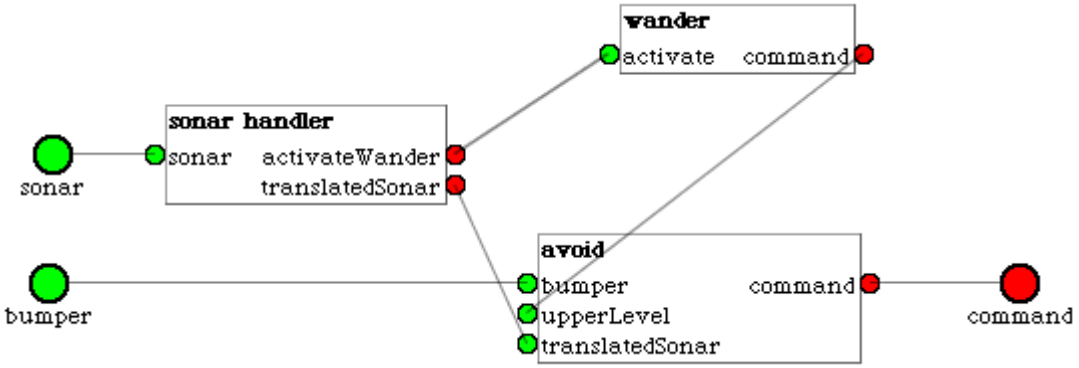
Obrázek 14 Stavový diagram komponenty Bumper Handler

Celkové zapojení komponenty *Avoid* v DEVS modelu vidíte na následujícím obrázku (obrázek 15). Vstupy zvnějšku tvoří jednak datové kanály senzorů – sonarů předzpracovaných *Sonar Handlerem* a bumperů – a navíc propojení, kterým mohou přicházet řídící příkazy z vyšších vrstev (které budou buď předány k vykonání, nebo zahozeny – dle stavu *Avoider*u). Výstup potom tvoří pouze příkazy ovládající pohyb robota.



Obrázek 15 Zapojení komponenty Avoid

Zapojení komponent ovládajících pohyb robota s porty vedoucími k rozhraní na robotický server *Player* je potom na obrázku 16. Z rozhraní přicházejí jako vstupy hodnoty o stavu senzorů a je na něj možno posílat příkazy ovládající pohyb robota.



Obrázek 16 Zapojení komponent realizujících řízení pohybu robota

4.3.3.2 Komunikační vrstva

Návrh vrstvy starající se o komunikaci mezi roboty přímo vychází z koncepcí chování a komunikačního protokolu v jednotlivých stavech (fázích) systému, které byly popsány v části 4.3.2.

V průběhu implementace a experimentování s touto vrstvou (nyní již experimenty samozřejmě probíhaly při zapojení obou robotů do simulovaného systému) se ukázalo jako výhodné modelovat ji jako dvě, do značné míry nezávislé komponenty. První – *Message Handler*, která udržuje nejvýznamnější složku stavu systému – tedy zda je robot autonomní, *Master* či *Slave* – a zpracovává příchozí symbolické zprávy. A druhou – *Distance Handler*, která se stará o všechny úkony zahrnující výměnu a porovnávání dat ze sonarů a iniciaci akcí z toho vyplívajících (utváření, udržování a

rozpouštění týmu). Iniciace akcí je právě jediným momentem, kdy dochází k interakci těchto komponent, přesněji *Distance Handler* upozorňuje na události komponentu *Message Handler*.

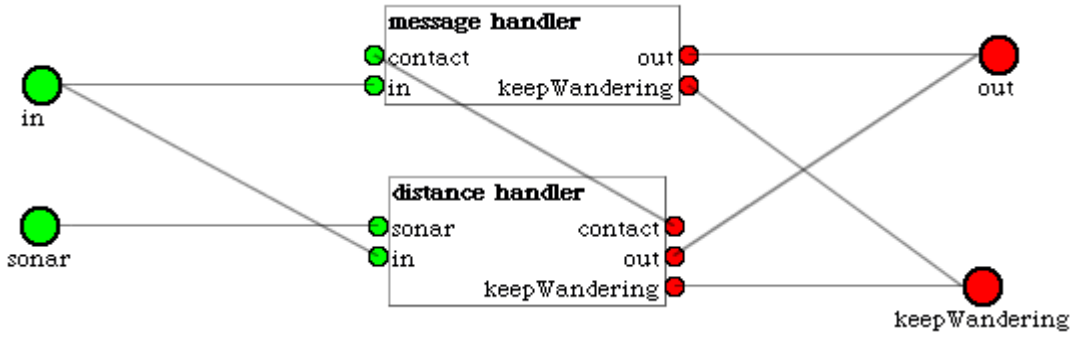
Distance Handler má tedy na starosti neustálý proces výměny a porovnávání dat ze sonarů. Efekt tohoto procesu se liší podle toho, ve které fázi se systém nachází. Pokud *Distance Handler* detekoval kontakt ve fázi hledání, sám iniciuje jeho potvrzování a obstarává průběh tohoto potvrzování. Až v momentě, kdy byl kontakt potvrzen, upozorní komponentu *Message Handler* na tento fakt zprávou *#confirmed*, která zahájí proces tvorby týmu. V případě, že kontakt potvrzen nebyl, pokračuje fáze hledání. Při vytváření týmu se *Distance Handler* stará o vyrovnání obou robotů, přesněji kontroluje informace ze sonarů druhého robota a čeká na moment, kdy jsou oba natočeni stejným směrem. Pokud takový moment nastane, je na to *Message Handler* upozorněn zprávou *#aligned* a začíná společný pochod. Pokud se, na druhou stranu, vyrovnání v časovém limitu nepodaří, je tento fakt oznámen v podobě zprávy *#lost*. Vše je potom opět navráceno do fáze hledání.

Důležitou zprávou zvnějšku, na kterou *Distance Handler* reaguje je *#sendDistances*, která funguje jako příkaz k zasílání dat ze senzorů. V případě, že daný robot řídí tvorbu týmu či ho vede, je ale zpráva vracena zpět. Sám má na starosti jen zpracování příchozích dat.

Jak bylo řečeno, *Message Handler* reaguje na příchozí zprávy a zajišťuje jejich realizaci. Množina zpráv, na kterou reaguje, pokud je v autonomním stavu, vypadá takto:

- *#stop* – robot zastaví a pošle zpět potvrzení o vykonané akci (*#ack*).
- *#move* – robot se rozjede a zašle o tom zpět potvrzení.
- *#enslave* – robot provede strukturální změny ve svém modelu řízení (přejde tím do stavu *Slave*) a následně se začne otáčet na místě, aby umožnil vyrovnání orientace obou robotů. Současně posílá zpět zprávu *#ack* jako potvrzení o vykonání celé akce.

V případě, že je komponenta ve stavu *Slave*, reaguje na jedinou zprávu - *#regainAutonomy*. Tato způsobí převod komponenty do stavu *Autonomous* a provede s tím spojené strukturální změny.

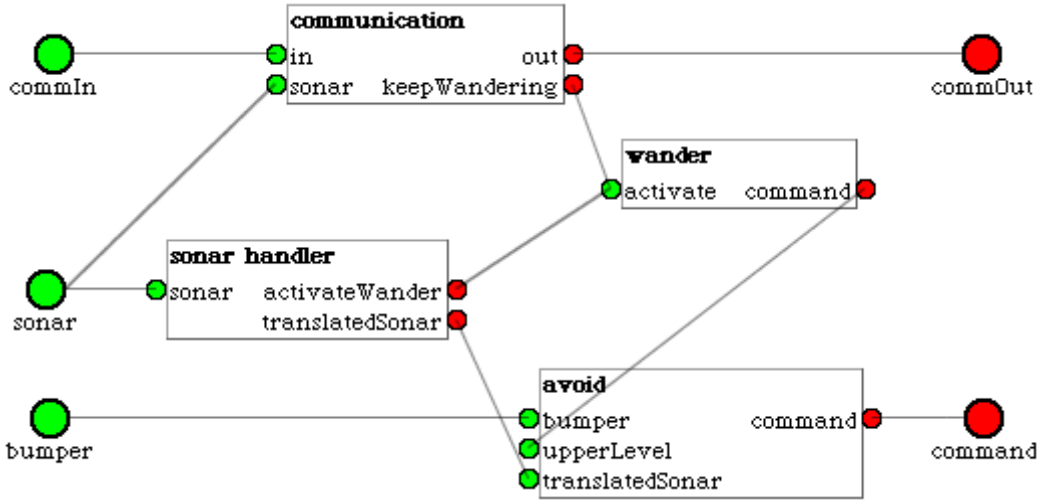


Obrázek 17 Propojení komponent v komunikační vrstvě.

Komponenty tedy pracují odděleně, z čehož vyplývá jejich paralelní uspořádání (obrázek 17). Jediný moment, kdy úzce spolupracují, je fáze vytváření týmu, ve které každá komponenta obstarává část komunikace.

Kvůli jejich rozsáhlosti jsou stavové diagramy obou komponent (*Message Handleru* a *Distance Handleru*) k nalezení na konci práce mezi přílohami.

Komunikační vrstva je zapojena do celkového modelu, který robota řídí tak, aby mohla robota zastavit či rozjet. Je tedy propojena s komponentou *Wander*. Navíc je napojena na komunikační kanál (vstup a výstup) a na port, kterým přicházejí aktualizace ze sonarů. Celé propojení vidíte na obrázku 18.



Obrázek 18 Celkové zapojení řídicího systému robota v subsumpční architektuře.

4.3.3.3 Strukturální změny

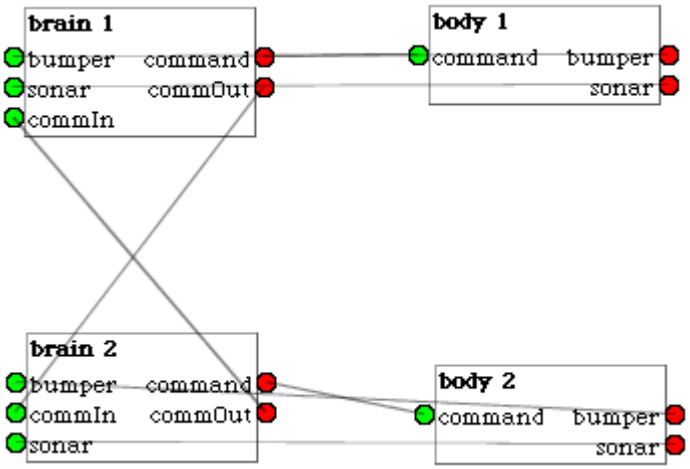
Při přechodu robota z autonomního stavu do stavu *Master* či *Slave* je nutné provést určité změny v modelu, které tuto roli reflektují a které zajistí očekávané chování modelu.

Při přechodu do role následovníka, je zrušeno propojení z komponenty *Avoid*, přes kterou jdou všechny příkazy řídící pohyb, na výstup do rozhraní na robotický simulátor. Místo něho je vytvořeno propojení z komunikační komponenty, která pouze přeposílá příkazy přijaté od vůdce. Při rozdělení týmu jsou provedeny reverzní změny v modelu – komplementární akce v opačném pořadí.

Při přechodu do role vůdce je vytvořeno propojení z komponent řídících pohyb do komunikačního kanálu – všechny příkazy, které bude vůdce vykonávat, jsou tak posílány následovníkovi.

4.3.3.4 Model celého systému

Podoba a tedy propojení řídicích systémů robotů a rozhraní na server pro řízení robotů Player je realizováno přesné tak, jak bylo nastíněno v konceptuálním návrhu (kapitola 4.3.1). Kontroléry tak reprezentují mozky a rozhraní těla robota. Podoba výsledného DEVS modelu je zachycena na obrázku 19.



Obrázek 19 Celkové propojení systému řešícího úlohu formování týmu subsumpční architekturou.

4.4 Řešení BDI architekturou

V této části je popsáno řešení úlohy využívající systému PNagent (viz. 3.2) – BDI architektury (viz. 2.1.3) postavené na Objektově Orientovaných Petriho Sítích jazyka PNtalk (viz. 3.2.1). Popisované řešení bylo přebráno od Ing. Zdeňka Mazala, v jehož disertační práci bylo použito jako demonstrační úloha systému PNagent. Pro větší přehlednost bylo mírně upraveno a pročištěno.

Tím, že řešení vznikalo za jiným účelem a v jiném kontextu než tato práce, jsou jeho východiska také poněkud odlišná. Nicméně zadání úlohy je stejné a výsledek je tedy plnohodnotným protějškem implementace subsumpční architekturou, se kterým bude srovnáváno.

4.4.1 Specifická východiska řešení

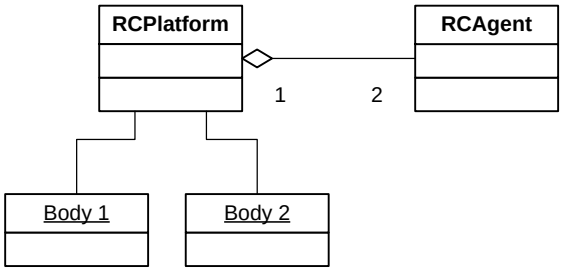
Při návrhu systému byl kladen důraz na využití zavedených a standardizovaných koncepcí využívaných v oblasti agentních a multiagentních systémů. Konkrétně je potom poukazováno na komunikační jazyk ACL (Agent Communication Language), jehož podmnožinu roboti ke komunikaci využívají, a prvky architektury agentních systémů. Obojí vychází ze standardů organizace FIPA[15] (Foundation for Intelligent Physical Agents).

4.4.2 Konceptuální model systému

Nejvýraznějším využitým prvkem, zavedeným v oblasti multiagentních systémů, je tzv. platforma. Na této platformě agenti existují a využívají jejích služeb (26). V tomto případě pro oba agenty platforma tvoří vrstvu, která zajišťuje převod surových dat ze senzorů na symbolickou reprezentaci, Data přicházejí z rozhraní na robotický server Player (i zde s využitím simulátoru Stage), který tak fakticky tvoří tělo robota. Symbolickou reprezentací dat jsou události, které jsou zpracovávány cyklem interpretace agenta – mají za následek vytváření plánů, jejich spouštění, pokračování, ukončování apod. Projevem vykonávání plánů může být pokyn k pohybu, který je od agenta, přes

[15] http://www.fipa.org

platformu poslán k vykonání tělu robota. Vztah platformy, agentů a objektů zapouzdřujících rozhraní na Player (*Body 1* a *2*) je znázorněno na obr. 20.



Obrázek 20 Model systému jako platformy obsahující agenty, kteří skrze ní přistupují řídí pohyb robotů.

Jednotlivé fáze systému zde reprezentují různé plány agentů, které jsou v dané chvíli zpracovávány. Plány, resp. jejich šablony, mají definovány události, v reakci na které se instanciují, zařadí se tak do intenční struktury a budou dále vykonávány.

Platforma také dokáže doručovat ACL zprávy adresátům v nich uvedeným – je jí proto využito jako komunikačního prostředku mezi agenty.

4.4.3 Chování a komunikační protokol

V této části bude popsána realizace jednotlivých fází, kterými systém – tedy především agenti řídící roboty – prochází. Fáze jsou v BDI architektuře identifikovatelné podle v dané chvíli aktivních plánů. Následující jednotlivé části se tak soustřeďují především na popis jednotlivých plánů, jejich spouštěcích událostí a vzájemné návaznosti.

Je důležité zmínit, že agenti i jejich plány využívají pro matematické a jiné nízko-úrovňové operace nad daty ze senzorů třídu v jazyce Smalltalk pojmenovanou *RCMath*. Dle slov autora, daná verze PNtalku nepodporovala některé potřebné konstrukce, proto byla realizována tímto způsobem.

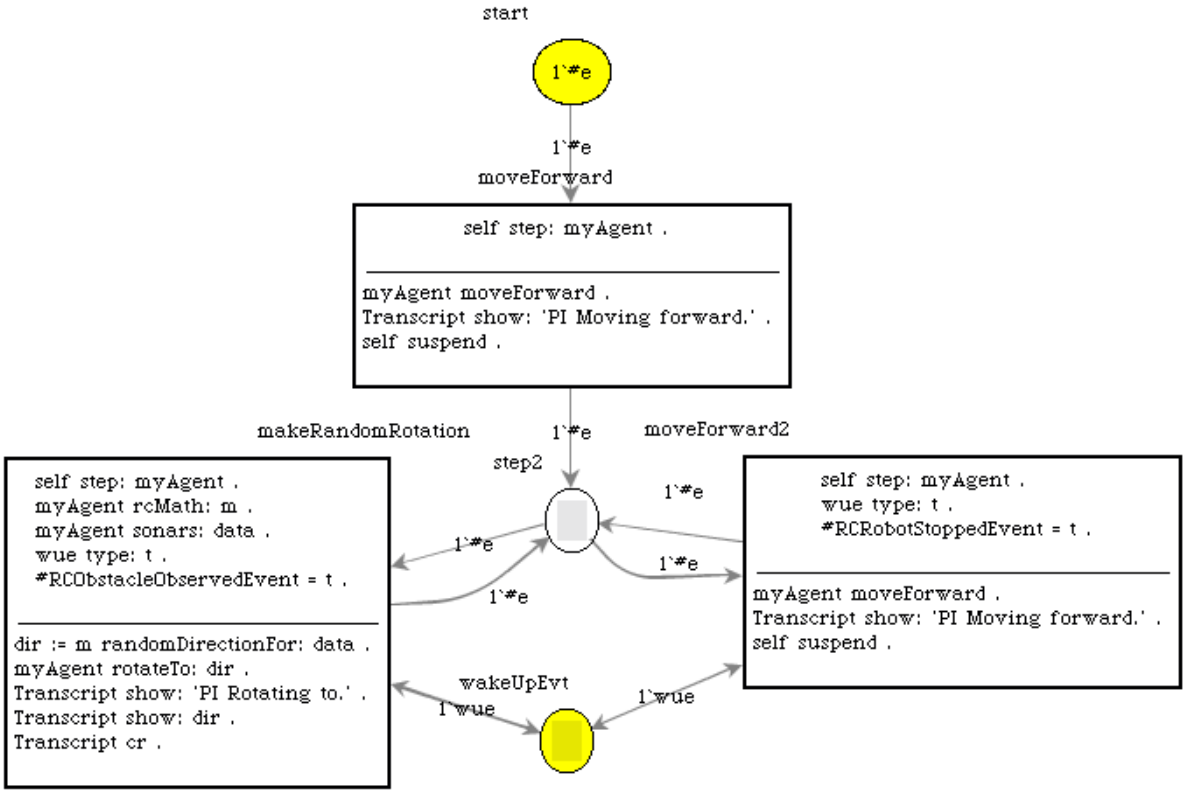
4.4.3.1 Fáze hledání kontaktu

Pohyb je zde realizován jako náhodný. Zjednodušeně řečeno, pokud je v kterémkoliv směru detekována překážka, robot se otočí do směru jednoho ze sonarů, které ukazují nejvyšší vzdálenost. V momentě kdy robot zastaví, tedy i otáčení, je aktivován pohyb vpřed.

Tento náhodný pohyb je realizován plánem *RCRandomMovePlanInstance*, který je vytvořen ze své šablony, pokud přijde událost typu *RCRobotStoppedEvent* a pokud ještě takový plán neexistuje (zabraňuje se tím nesmyslné duplicitě). Po vytvoření plánu je prvním krokem plánu rozjetí robota vpřed. Poté robot reaguje dle příchozích událostí.

Pokud je na sonarech zaznamenána překážka, kterážto situace je reprezentována událostí typu *RCObstacleObservedEvent*, je robot otočen ve směru sonaru, na kterém je největší naměřená hodnota (pokud sonarů s maximální hodnotou je více, je jeden z nich vybrán náhodně). Na druhou

stranu pokud byl robot zastaven (již zmíněná *RCRobotStoppedEvent*) uvede plán robota do pohybu vpřed.



Obrázek 21 Vizualizace plánu RCRandomMovePlanInstance v uživatelském rozhraní prostředí PNtalk.

Pro představu, jak vypadá plán vizualizovaný v uživatelském rozhraní prostředí PNtalk, je tento zachycen na obrázku č. 21. U následujících fází vizualizace plánů není uvedeno – při jejich velikosti není možné je rozumně zachytit a umístit v textu ani do příloh.

4.4.3.2 Fáze potvrzování kontaktu a tvorby týmu

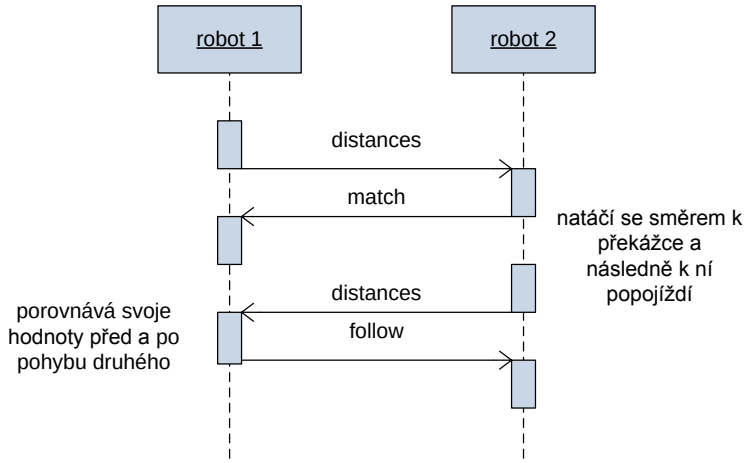
U robota je v případě spatření překážky, vytvořena instance plánu *RCFindFollowerPlanInstance*, která má za úkol zahájit komunikaci s druhým robotem za účelem potvrzení či vyvrácení vzájemného kontaktu.

V první fázi plán zastaví robota a zašle druhému robotovi zprávu s aktuálními hodnotami svých sonarů. V reakci na tuto zprávu je vytvořen u druhého robota plán *RCReplyFFPlanInstance* – přijetí zprávy je v jeho šabloně[16] definováno jako spouštěcí událost. Ten robota zastaví a pokusí se najít shodu mezi svými a přijatými vzdálenostmi. Pokud shodu nenajde – překážka, kterou spatřil vyzývající robot není druhý robot – odpovídá tento zprávou s obsahem *#noMatch* a oba plány tím končí – roboti se vracejí zpět do fáze hledání.

[16] Objekt, který pouze kontroluje spustitelnost plánu a případně vytváří instanci, která ho skutečně implementuje, a která je následně přidána mezi aktivní plány agenta.

V případě shody se odpoví indexem shodného sonaru a robot se současně natočí tak, aby směřoval čelem tam, kam původně směřoval tento sonar. Následně se začne pohybovat vpřed, dokud se k překážce, potenciálně druhému robotovi, nepřiblíží hodně blízko (událost typu *RCObstacleDeadAheadEvent*). V reakci na ní zašle vyzývajícímu robotovi zprávu s hodnotami svých sonarů, která slouží jako oznámení o dokončení potvrzovacího pohybu. Vyzývající robot (s plánem *RCFindFollowerPlanInstance*) porovná hodnoty svých aktuálních sonarů s těmi před potvrzovacím pohybem. Pokud se změnily, zasílá zprávu *#follow*, v opačném případě *#dontFollow*. Tato část protokolu, v úspěšné variantě, je ve formě diagramu sekvence jazyka UML na obrázku 22.

Na úspěšné potvrzení ještě robot reaguje natočením se do stejného směru jako nynější vůdce a potvrzením tohoto pohybu. Před ukončením plánu si vůdce u sebe vyvolá interní událost *RCLeadGoal* a následovník *RCFollowGoal*. Tyto mají za následek instanciaci plánů umožňujících společný pochod.



Obrázek 22 Komunikační sekvence potvrzování kontaktu v úspěšné variantě.

Důležité zmínit, že je zajištěno, že robot může mít aktivován buď plán na vyhledávání, nebo na odpovídání – nikdy oba zároveň. Stejný mechanismus zabrání i duplicitním plánům. Implementován je jako místo v objektové sítí instance agenta (instance třídy *RCAgent*) pojmenované *findFollowerPlan*. Šablony plánů *RCFindFollowerPlanTemplate* a *RCReplyFFPlanTemplate* kontrolují, zda je toto místo prázdné. Pokud je, vloží do něj značku a vytvoří instanci implementující daný plán. Pokusy o spuštění konkurenčních plánů tak selžou.

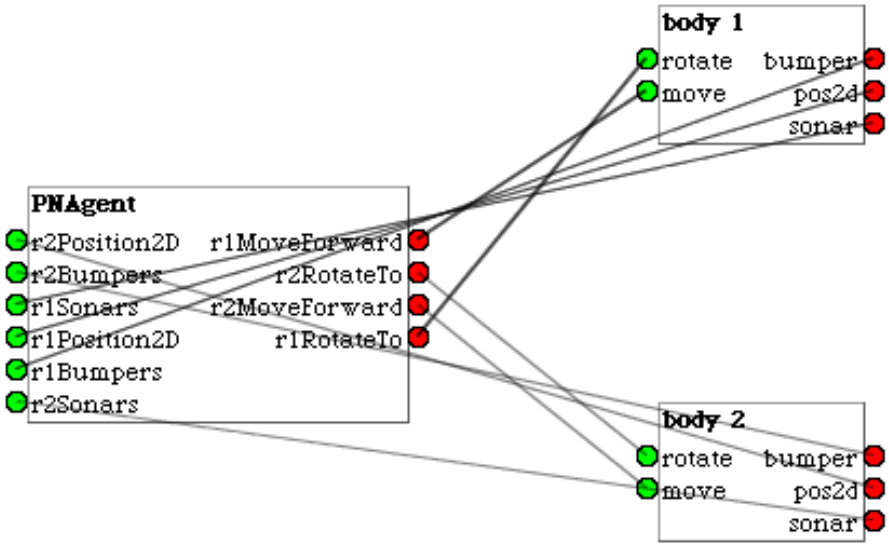
4.4.3.3 Fáze společného pochodu

Plány na společný pohyb (*RCLeadPlan* a *RCFollowPlan*) jsou navrženy tak, aby nahradily plány, které vytvoření týmu realizovaly (ty také při svém úspěšném ukončení odstraní značku z místa *findFollowerPlan* a plány pro vedení resp. následování je tak můžou nahradit). *RCLeadPlan* i *RCFollowPlan* jsou de facto variací na plán realizující náhodný pohyb. Plán vůdce oproti němu při každém pohybu zasílá následovníkovi zprávu, definující tento pohyb. Plán následovníka tyto zprávy přijímá a příslušné pohyby vykonává.

V případě, že se robot s plánem *RCFollowPlan* příliš přiblíží k nějaké překážce (zpráva typu *RCObstacleDeadAheadEvent*), informuje o tom vůdce a tento plán ukončí. Následovník v tu chvíli totiž nemůže pokračovat ve vykonávání příkazů bez toho, aby narazil. Vůdce po přijetí zprávy informující o rozpuštění týmu svůj plán také ukončí a celý systém se tím vrací do fáze hledání kontaktu.

4.4.4 Popis výsledného modelu

Ve chvíli, kdy máme platformu s agenty – obojí popsáno v jazyce PNtalk, je nutno jí, a s ní agenty, napojit na rozhraní na robotický server Player, který je realizován jako sdružený DEVS model. Propojení těchto komponent, modelovaných různými paradigmaty, je realizováno obalením PNtalk objektu wrapperem[17], jak bylo popsáno v kapitole 3.2.1. Výsledná podoba tohoto modelu je zachycena na obrázku 23.



Obrázek 23 Pohled na propojení platformy s BDI agenty a těly s rozhraními na simulátor Player/Stage.

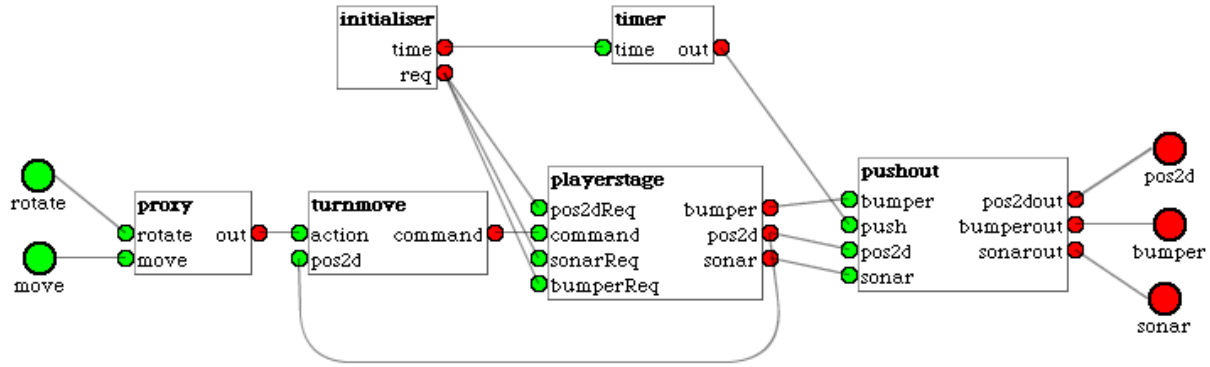
4.4.4.1 Komponenty realizující pohyb

Platforma přijímá surová data od Playeru a převádí je na symbolickou reprezentaci. Převod příkazů ze symbolické podoby na vykonavatelnou ale nechává komponentám ve formalismu DEVS umístěným mimo ní. Součástí těla, které obsahuje především rozhraní na Player, jsou z toho důvodu i komponenty zajišťující a řídící pohyb.

U příkazů pro pohyb vpřed a zastavení je použit jednoduchý překlad z hodnot na příslušné příkazy, které rozhraní na Player (komponenta *playerstage* na obrázku 24) vykoná. Je ale nutné realizovat příkazy k rotaci, jejichž parametrem je úhel v radiánech, o který se má robot pootočit. Tento pohyb je možno implementovat díky informaci o relativním natočení robota vzhledem k výchozí pozici (získatelného z odometrie, viz kapitola 4.2.1). Jednoduše je spočítáno cílové natočení a robot se

[17] Odvozeno od návrhového vzoru Wrapper, jehož je popisovaná konstrukce konkrétní realizací.

začne otáčet okolo své osy. Na základě aktualizací dat o poloze robot zastaví, pokud aktuální poloha s jistou malou odchylkou odpovídá té požadované. Tyto akce jsou implementovány komponentami *proxy* a *turnove*.



Obrázek 24 Uspořádání těla agenta v PNagentu.

Tělo také obsahuje komponentu *pushout*, která dodává platformě aktualizace ze senzorů v daných časových intervalech. To je vynuceno výpočetní náročnosti – tedy pomalostí – jazyka PNtalk. Při frekvenci aktualizací, se kterou je dodává Player by mohlo dojít k zahlcení a potenciálně tak k nekorektnímu chování.

5 Srovnání modelů a zhodnocení

Tato kapitola je věnována porovnání vlastností realizovaných řídicích systémů robotů – tedy systému implementovaného přímo ve SmallDEVS s využitím subsumpční architektury a systému využívajícího PNtalk a BDI architekturu. Pozorování týkající se modelů samozřejmě vycházejí z uvedené dokumentace. Zároveň s nimi bude uvedeno několik úvah zobecňujících pozorovaná fakta jak z hlediska implementačních platforem, tak použitých agentních architektur, ale především z hlediska vývoje metodikou zdola-nahoru. Postupně se budeme věnovat různým kritériím, která je možno považovat za důležitá při hodnocení jakéhokoliv systému, ale systému s komplexním chováním obzvláště. Na závěr bude vše shrnuto do několika tezí.

5.1 Srozumitelnost modelu

Srozumitelností modelu je myšlen čas nutný k pochopení logiky a odhalení detailů fungování modelu při znalosti formalismů a přístupů – souhrnně bychom mohli říci východisek, se kterými byl tento model vytvořen. Tento aspekt je důležitý při tvorbě systému inkrementálním modelováním, protože usnadňuje analýzu a interpretaci výsledků iterací vývoje systému a usnadňuje tak jeho další úpravy. Také přispívá ke zjednodušenému hledání chyb, ulehčuje vývoje v týmech, činí systém méně náročným na dokumentaci apod.

U modelu vytvořeného formalismem DEVS, s využitím nadstavby konečných automatů a principů subsumpční architektury, je srozumitelnost prokazatelně nižší než u BDI modelu. K pochopení výstupů modelu jako celku a kauzality procesů uvnitř něj je totiž často nutné analyzovat několik komponent zároveň (např. v případě komunikační vrstvy, jejíž chování je dáno kompozicí dvou paralelně zapojených komponent). Funkcionalita určená k dosažení nějakého cíle je tedy distribuována mezi více prvků systému. Navíc i jedna komponenta může být určena k dosažení více cílů (komunikační vrstva se stará o detekci kontaktu, vytváření týmu i jeho rozpad). U implementace v PNagentu je na druhou stranu popis každého plánu, tedy určité sekvence cíleného chování, koncentrován na jednom místě – v objektu jazyka PNtalk, který tento plán implementuje. Při správném návrhu je možno, a dokonce žádoucí, dosáhnout i stavu, kdy jeden nehierarchický plán plní právě jeden cíl. Mapování cílů na plány je tedy možno, za ideálního stavu, nazvat injektivním, čehož je v případě subsumpční architektury a mapování na komponenty prakticky nemožné dosáhnout. Analýza výsledného chování může být samozřejmě komplikovanější při více aktivních plánech zároveň, ale výhoda popisu chování vedoucího k dosažení jednoho cíle na jednom místě přetrvává.

Funkcionalita atomického modelu DEVS je navíc distribuována mezi více metod (interní a externí přechodová funkce, případně vlastní pomocné metody), takže důkladná analýza posloupností událostí i v rámci atomického modelu může být relativně zdlouhavý proces. To omezuje i implementaci konečných automatů, pokud modelujeme komponenty v těchto intencích, kde je nutno pečlivě rozdělovat přechody automatu mezi interní a externí přechody DEVS modelu. Obecně u objektů jazyka PNtalk je také funkcionalita rozdělena mezi síť objektu a sítě metod, ale díky

možnosti vizualizace Petriho sítí, využití inskripčního jazyka a v neposlední řadě při dodržování korektního objektového návrhu, se zdá celkový účel takových objektů pochopitelnější.

Tato vlastnost má, jak bylo řečeno vliv i na snadnost dokumentace systému. Určité závěry lze vyvodit i z rozsahu dokumentačních částí věnovaných jednotlivým modelům (kapitoly 4.3 a 4.4). I při vzetí do úvahy, že jeden model byl tvořen a druhý jen upraven a zdokumentován, rozdíl v rozsahu je poměrně markantní. Je to dáno především poměrně dlouhou cestou od navrhovaného modelu komunikace a komponent k faktické implementaci modely DEVS. Na cestě od jednoho k druhému je nutno provést řadu transformací a hodně faktů tak muselo být popsáno dvakrát, ale na různých úrovních abstrakce, aby byly zahrnuty všechny relevantní informace. Nezahrnutí detailů implementace nebo výchozích koncepcí by značně znesnadnilo pochopení systému. U systému popsaného Petriho sítěmi tato nutnost odpadla – zvláště ve vizualizované podobě, ale i v textové (popisné), jsou koncepce a jednotlivé kroky komunikace zřejmé – konceptuální návrh a implementace zde splývají. Je to zcela zřejmě dáno rozdílnou úrovní abstrakce, se kterou pracuje DEVS a Objektově Orientované Petriho Sítě.

5.2 Rozšiřitelnost

Rozšiřitelností je myšlena obtížnost přidání další funkcionality do systému, daná mírou vynaloženého úsilí. V případě agentů je funkcionalitou zřejmě nová množina chování, zpřístupňující plnění nových cílů. Je to zásadní vlastnost při tvorbě modelů metodologií zdola-nahoru a při inkrementálním a interaktivním modelování obzvláště.

S určitou rezervou hodnocení této vlastnosti v daných modelech odráží hodnocení u srozumitelnosti modelu. I zde velkou roli hraje míra rozptýlenosti funkcionality mezi různé komponenty systému. U subsumpční architektury míra vynaloženého úsilí daleko vyšší a může tedy být složité, zvláště u komplexních modelů, bezpečně a efektivně implantovat nová chování do stávající struktury bez narušení korektní funkcionality a tím implikovaných dalších úprav. Výsledné chování je totiž tvořeno souhrou mnoha komponent a neúmyslné narušení může mít fatální dopady na celkovou korektnost.

Naproti tomu u agenta v PNagentu je rozšíření spektra chování, alespoň u modelu realizujícího vybranou úlohu – bez použití hierarchických plánů a výskytu několika současně aktivovaných plánů, otázkou přidání nových událostí a s nimi souvisejících plánů. Rozšíření je tedy možno považovat za poměrně čisté, intuitivní a relativně bezpečné. Při využití hierarchických plánů či při výskytu situací s několika současně aktivovanými plány se stejnými prioritami by mohl nastat problém, např. s jejich konfliktními chováními. Hlubší analýza této problematiky ale překračuje zaměření této práce i znalosti autora.

5.3 Výpočetní efektivita

Výpočetní efektivita je pro fyzické agenty – roboty – pohybující se v reálném světě poměrně zásadní. Čím efektivnější vzhledem k využití výpočetních zdrojů model řídící robota je, tím sofistikovanější procesy mohou na stejném hardwaru běžet bez ztráty vyžadované reaktivity.

Subsumpční architektura jako taková je, díky filosofii decentralizace, snadno paralelizovatelná, potažmo efektivně implementovatelná v hardwaru. Jednotlivé komponenty jsou na sobě nezávislé a mohou být tedy spuštěny současně (např. paralelní zpracování dat ze sonarů v komunikační vrstvě a v *Sonar Handleru*).

Systém SmallDEVS běžící ve Squeak Smalltalku je, na druhou stranu, jako takový komparativně pomalejší vzhledem k rychlosti systémů v jiných jazycích, i takových které využívají ke svému běhu virtuální stroj. Je to dáno kromě jiného i celkově dynamickou (míněno ve smyslu pozdní vazby) povahou samotného Smalltalku a také samozřejmě formální bází SmallDEVSu. Navíc, kvůli stávající architektuře nedokáže Squeak využít potenciálu více-jádrových procesorů, protože celý běží v jednom vlákně hostitelského operačního systému (27).

Běh BDI modelu je na tom ale ještě hůře, interpretace sítí PNtalku je evidentně velice pomalá a byla proto nutnost využít i konstrukce taktované zpřístupňování aktualizací senzorů, bez níž by, při frekvenci aktualizací ze serveru *Player*, mohlo dojít k zahlcení. Tento přístup u modelu se Subsumpční architekturou nebyl potřeba. Ani s hypotetickou paralelizovatelností, alespoň v případě robotů, na tom není celé řešení nejlépe – i přesto, že Petriho sítím je tato vlastnost inherentní. Úzkým hrdlem je totiž v případě frameworku PNagent cyklus interpretace plánů – kde je možno v každou chvíli vykonávat jen jeden plán. V případě softwarových agentů by bylo možno uvažovat paralelní interpretaci, v případě fyzických agentů jsme ovšem omezeni tělem robota. V PNagentu je alespoň cyklus interpretace rozdělen do dvou nezávislých částí – zpracování událostí a zpracování plánů. Určitý prostor pro paralelizaci tu tedy je.

5.4 Distribuovatelnost

Koncept distribuovatelnosti úzce souvisí s výpočetní efektivitou – pokud různé části modelu běží na různých výpočetních systémech, tedy paralelně, může být celý výpočetně efektivnější z hlediska času (záleží na konkrétní úloze). Jsou ovšem modely, kde taková potřeba je inherentní řešenému problému – příkladem můžou být senzorové sítě, kde lze také uvažovat využití agentů – např. (28). Pokud platforma, ve které model vyvíjíme, distribuovatelnost podporuje, je možné systém namodelovat na jednom stroji a bez jakéhokoliv úsilí ho následně spustit v distribuovaném prostředí, což celý vývoj značně zjednodušuje.

Dále se pokusíme zhodnotit, jak obtížné může být realizovat distribuovanou simulaci výsledných modelů, přičemž princip dekompozice je jasný – každý robot bude řízen jiným výpočetním uzlem.

V případě DEVS modelu je distribuovatelnost možno zařídit dvěma způsoby. První z nich je do jisté míry realizovatelný již nyní – realizace komunikačního kanálu pomocí prostředků mimo simulovaný model – např. síťově spojení mezi simulacemi nebo komunikace přímo skrz prostředí, ve kterém se roboti pohybují (např. využití zvukových signálů). Při spuštění takového systému distribuovaně je negativem z formálního hlediska faktická existence dvou simulací, což je pro analýzu evoluce systému poněkud nepraktické a navíc porušující koncept kontinuity modelu. Ale z hlediska běhu systému je takové řešení bezproblémové a zcela přijatelné. Druhým způsobem je potom

implementace distribuovaného simulátoru DEVS modelů do SmallDEVSu, který by tak umožnil koordinovanou simulaci probíhající na více výpočetních uzlech – za tohoto předpokladu by bylo možné používat ke komunikaci i stávající propojení modelů.

U modelu realizovaného BDI architekturou je situace komplikována použitím konceptu platformy, zavedeného v multiagentních systémech. Pokud by byla snaha zachovat platformu jako analogii světa, ve kterém se roboti pohybují, bylo by nutné implementovat schopnost platformy běžet distribuovaně na více výpočetních uzlech – tedy zavedení distribuovatelnosti na úrovni této třídy objektů do systému PNagent. V tomto případě, kdy agenti nesdílí pomocí platformy žádné informace (stav prostředí je definován mimo ní), pouze skrze ni komunikují, si to lze poměrně dobře představit.

Druhou možností by byla podpora distribuované exekuce objektů přímo do systému PNtalk, což by pravděpodobně bylo značně náročné. Jako nejlepší se rozhodně jeví varianta implementace platformy jako DEVS modelu, ne jako PNtalk objektu, které jsou stejně obaleny a umístěny do SmallDEVS simulace. Problém se tak totiž redukuje na problém distribuovatelnosti modelů ve formalismu DEVS resp. na podporu distribuované simulace ve SmallDEVS. Toto řešení je zdaleka nejuniverzálnější.

5.5 Vývoj v simulaci a podpůrné nástroje

V této části se podíváme na skutečnou podporu vývoje v simulaci a obecně nástroje, které je možno ve SmallDEVS a PNtalku využít při tvorbě modelů a jejich simulaci.

Nutno předeslat, že SmallDEVS je na tom neporovnatelně lépe. Obsahuje celkem intuitivní rozhraní pro vizualizaci a editaci atomických i sdružených modelů. Drobnou komplikací je snížená rychlost odezvy u komplexních modelů, ale není to nic, co by práci výrazně komplikovalo. Při práci s výrazně hierarchickými modely je nutno využít nástroje MyRepository, který model zobrazuje ve stromové struktuře.

SmallDEVS podporuje úpravy během simulace, a to jak strukturální – změny propojení, tak úpravy kódu metod. Strukturálních změn také bylo v implementovaném modelu využito. Není ani problém simulaci pozastavit a následně ji opět spustit[18] – toto musí ovšem být reflektováno i v tvořeném modelu. V případě vytvořené implementace musela být po znovuspuštění provedena explicitní změna stavu komponenty *Sonar Handler*, aby se dal robot opět do pohybu, když byl při pozastavení simulace zastaven.

Situace u systému PNtalk je značně problematičtější. Jeho rozhraní v současné verzi dokáže sítě zobrazit, ale úpravy přes toto rozhraní provedené se do kódu nepromítnou. Jedinou přístupnou variantou je tedy editace zdrojového kódu tříd, což je sice přijatelné, ale výhoda, kterou Petriho sítě mají – tedy existence ekvivalentní vizuální reprezentace – se tím stírá. Provedení úprav v průběhu simulace v PNtalku možné sice je, ale je extrémně riskantní, neboť zatím nejsou implementovány

[18] Toho skrytě využívá i SmallDEVS, pokud je během simulace z resp. do sdruženého modelu odebírán resp. přidáván jiný model.

mechanismy zaručující zachování konzistence při prováděných změnách. Nevhodná změna tak může upravovaný objekt fatálně poškodit.

5.6 Shrnutí

Z úvah uvedených u jednotlivých důležitých vlastností modelů, potažmo platforem, na kterých byly vytvořeny, je možné vyvodit obecnější závěry pro vývoj kontrolérů robotů a inteligentních agentů obecně.

Systém SmallDEVS není, kvůli své relativně nízké úrovni abstrakce, obecně vhodný pro popis nadměrně komplexních systémů – ačkoliv je to teoreticky samozřejmě možné. Rychlost simulace modelů je s jistými výhradami přijatelná pro vývoj a experimentování a v určitých případech by stačila i v reálném nasazení. Implementace distribuované simulace by jeho použitelnost v tomto ohledu značně rozšířila, nehledě na to, že by zpřístupnila SmallDEVSu třídu systémů, které distribuovatelnost vyžadují.

Ve spojení se subsumpční architekturou, jakožto obecně paralelním, distribuovaným a především reaktivním přístupem k návrhu agentů, tvoří pro návrh jednodušších agentů, i těch pohybujících se v přirozeném světě, poměrně silný nástroj. Jednodušších proto, že za určitou hranicí komplexnosti je srozumitelnost a především další rozšiřitelnost značně komplikována decentralizovanou povahou řízení. Použitelnost pro inkrementální vývoj zdola-nahoru tedy klesá s rostoucí komplexností modelu

PNagent je naproti tomu teoreticky opravdu mocný nástroj pro popis komplexních agentů. Výhodou BDI architektury jako takové je modelování agentů v intencích lidského uvažování o jiných inteligentních stvořeních. Ve spojení s vysokoúrovňovým jazykem s vizuální notací a s formálním základem v Petriho sítích je PNagent jasnou volbou pro návrh agentů, kteří musí disponovat sofistikovaným a dlouhodobě koordinovaným chováním. Množinu chování je možno i poměrně čistě a jednoduše rozšiřovat, pokud jsou spouštěcí podmínky plánů navrženy disjunktně. Inkrementální vývoj je zde tedy technicky velice snadno realizovatelný. V momentě kdy bude PNtalk umožňovat editaci modelů v simulaci, se stane tento způsob vývoje jednoznačně nejproduktivnější.

Zásadní limitací PNagentu je ovšem pomalost běhu, která ho reálně diskvalifikuje pro použití k řízení robotů v jakémkoliv dynamičtějším prostředí. Pro návrh softwarových agentů si lze představit třídu úloh, u kterých není vyžadována vysoká reaktivnost, ale zároveň je potřeba koordinace, plánování a dalších vlastností, které může PNagent nabídnout. V současné fázi vývoje, jak z hlediska jádra systému, tak z hlediska nástrojů pro práci s ním, je ale PNtalk, na kterém PNagent stojí, stále daleko před branou praktické použitelnosti.

6 Závěr

Úkolem práce bylo srovnat implementace vybrané robotické úlohy za účelem porovnání platforem a s nimi souvisejících agentních architektur z hlediska vhodnosti pro interaktivní a inkrementální modelování agentů resp. kontrolérů robotů. První srovnávanou platformou byl SmallDEVS – reflektivní a otevřená implementace formalismu DEVS v prostředí Squeak Smalltalk – v kombinaci s reaktivní subsumpční agentní architekturou. Druhou potom PNagent – implementace deliberativní BDI architektury v prostředí PNtalk, stojícím na konceptu Objektově Orientovaných Petriho Sítí.

Zvolena byla úloha *Formování týmu*, popisující distribuovaný robotický systém, která kromě řízení robota samotného obsahuje i prvek meziagentní komunikace a koordinace, díky čemuž se dá zařadit i mezi obecné multiagentní systémy. Tato kombinace aspektů vedla k volbě této konkrétní úlohy s důvěrou, že je dostatečně komplexní, aby se u ní projevily klady i zápory srovnávaných platforem.

Implementace na platformě SmallDEVS byla vytvořena od základů, implementace v systému PNagent byla potom s drobnými úpravám přebrána od Ing. Zdeňka Mazala. Obě implementace řeší stejnou úlohu s drobnými rozdíly ve výsledném chování robotů, s většími rozdíly v komunikačních mechanismech a s diametrálními rozdíly v použitých prostředcích.

Výsledné modely byly pečlivě zdokumentovány a bylo provedeno srovnání jejich vlastností v kontextu použitých technologií a z hlediska různých kritérií, která jsou podstatná pro vývoj kontrolérů fyzických agentů. Z těchto bylo dále vyvozeno několik obecnějších tezí shrnutých v následujících odstavcích.

SmallDEVS se subsumpční architekturou je dobrým nástrojem pro vývoj jednodušších reaktivních kontrolérů robotů i softwarových agentů. Jeho podpora pro vývoj v simulaci je dostatečná. Problém je ale s modely na principu subsumpční architektury, u nichž bezpečná rozšiřitelnost a tedy aplikovatelnost inkrementálního modelování klesá s komplexností upravovaného modelu poměrně rychle. Při případné implementaci distribuované simulace do SmallDEVS lze uvažovat i o reálném využití v rámci multiagentních systémů.

PNagent je naproti tomu teoreticky mocným nástrojem pro modelování komplexních, sofistikovaných a plánujících agentů na vysoké úrovni abstrakce. Z části díky Objektově Orientovaným Petriho Sítím, jakožto vysokoúrovňového formalismu a z části použitím BDI architektury, jejíž potenciál při vývoji metodologií zdola-nahoru se zdá být omezen jen kvalitami návrháře. Bohužel ale v současné době není dobře podporována editace modelů v simulaci, která by potenciál PNagentu dále zvýšila. Z použití v dynamických prostředích, kde by byla vyžadována vysoká reaktivita, je ale diskvalifikován svou pomalostí při vykonávání modelů. Pro skutečný vývoj se v současné době, díky své celkové nevyzrálosti, nedá v žádném případě doporučit.

6.1 Možné úpravy implementací úlohy

V první řadě lze uvažovat úpravy do samotné formulace úlohy - ty se samozřejmě projeví i do její implementace. Takovou modifikací je zvýšení počtu robotů v prostředí. S tím je potom spojeno několik variant řešení komunikace – buď zachování peer-to-peer modelu nebo zavedení centrálního prvku, přes který spolu budou roboti vzájemně komunikovat. V takovém případě by byly nutné změny v komunikačním protokolu implementace subsumpční architekturou, aby zprávy obsahovaly i adresáta a odesílatele. Další možnou změnou v zadání úlohy je odstranění předpokladu, že roboti jsou jediné pohybující se entity v prostředí. Tato vnáší do úlohy zvýšenou míru realismu a její řešení by vyžadovalo robustní a sofistikované metody. Otázkou je, zda toto zadání je skutečně řešitelné za použití stávajícího senzorického vybavení robota.

Nyní přejděme k dílčím úpravám realizovaných implementací. Řešení úlohy subsumpční architekturou lze v první řadě modifikovat změnou realizace stávajícího komunikačního kanálu, který má nyní podobu standardního DEVS propojení. Jednou možností je zapojení nesimulované části do simulace – lze si představit např. komunikační kanál reprezentovaný síťovým spojením. Tím by se implementace stala okamžitě distribuovaně spustitelnou. Další možností je realizace komunikace skrze prostředí, ve kterém se roboti pohybují – v úvahu by připadal například zvuk. Otázkou je, zda něco takového simulátor Stage či Gazebo dokáže. V opačném případě by k takovým experimentům byli nutní skuteční fyzičtí roboti.

V případě implementace PNagentem by bylo zajímavé pokusit se o implementaci platformy, která by byla distribuovatelná. V první fázi takovou, která by umožňovala doručování zpráv mezi svými částmi na různých výpočetních uzlech. Fakticky se tak jedná o síťový komunikační kanál jako v možné úpravě nastíněné pro SmallDEVS, ale ve shodě se zavedenými postupy v oblasti agentů – tedy na úrovni služeb agentům poskytovaných.

Velice zajímavou variantou nakonec je nahrazení komunikační vrstvy v implementaci subsumpční architekturou modelem v PNtalku – pro návrh a implementaci komunikačních protokolů mají Petriho sítě velice dobré vlastnosti a jsou k tomu často využívány. Každá úroveň řízení robota by tak byla řízena komponentou operující na adekvátní úrovni abstrakce.

6.2 SmallDEVS, PNtalk a vývoj obecně

Připomínky a návrhy směřované k prostředím a nástrojům SmallDEVS a PNtalk lze jen stěží formulovat čistě pro návrh kontrolérů robotů či inteligentních agentů, ačkoliv samozřejmě mají svá specifika. V současné fázi vývoje těchto platforem je, dle mého názoru, nejdříve nutné se soustředit na problémy a nedostatky, které jsou spojeny s vývojem složitých systému obecně. Jakákoliv specializace na jistou třídu systémů nedává v dané chvíli smysl.

U systému SmallDEVS jsou úvahy o něco konkrétnější. Obecně směřují spíše k uživatelskému rozhraní, než k samotnému simulačnímu systému, který se z používání jeví jako poměrně stabilní a spolehlivý (záleží samozřejmě na chování konkrétního modelu).

Z hlediska celkového zjednodušení orientace a navigace ve složitých modelech bych považoval za prospěšný browser[19] modelů s následujícími vlastnostmi:

- Možnost navigace v hierarchickém modelu v rámci jednoho otevřeného okna. Tedy umožnění zanořování se a vynořování se do resp. ze sdružených modelů.
- Možnost navigace mezi komponentami modelu po dráze toku informace. Tedy přes porty a na ně napojená propojení.
- Možnost inspekce stavu atomických modelů v kontextu sdruženého modelu, do kterého patří – tedy bez otevření nového okna. Tato řešení by značně usnadnilo sledování stavu modelu v simulaci.

Z podpůrných nástrojů pro vývoj a především ladění modelů by byl užitečný nástroj umožňující zobrazení záznamů událostí (logů) pouze vybraných subsystémů či komponent. Stejně tak možnost zobrazit pouze určité typy událostí (např. interní přechody a s nimi spojené výstupy komponent). Formální základ, umožňující zaznamenávání evoluce systému na nejnižší úrovni, k flexibilní práci s tímto záznamem přímo vybízí.

Co by SmallDEVS jako takový posunulo blíže k použitelnému model-base designu, je nástroj na vizuální návrh stavových automatů. Momentálně je možné v intencích automatů pracovat s nadstavbou nad DEVS modelem, ale je nutno ručně provádět dekompozici na interní a externí přechody. S nástroji pro práci s FSM ve vizuální doméně by bylo snadnější a přímější navržený systém převést do jeho spustitelné podoby. Automatizovala by se ona nutná manuální transformace z modelu do kódu. Tento nástroj by měl i přímý dopad na proces modelování agentů subsumpční architekturou.

U PNtalku je prvním nutným krokem skutečně použitelný nástroj pro práci se sítěmi a objekty. Nástroj, který bude poskytovat možnost prohlížet a editovat sítě tříd a nejlépe i zobrazovat stav sítí instancí tříd jazyka PNtalk. To vše samozřejmě v grafické podobě.

Dalším možným krokem, jehož nutnou podmínkou je vyzrálý PNtalk, je realizace podpůrný prostředků vývoje agentů v PNagentu. Primární je v takovém případě ladící nástroj šitý na míru struktuře BDI agentů. Takový debugger musí zobrazovat a umožnit inspekci příchozích událostí, momentální cíle, aktivní plány (obecněji záměry) a bázi představ. Za současného stavu totiž může být poměrně obtížné určit, kde je při nekorektním chování chyba.

[19] Ve Smalltalku užívaný název pro nástroj pro procházení množinou entit a jejich editaci. Českému ekvivalentu „prohlížeč“ většinou v běžném chápání významu není přisuzována schopnost modifikace prohlíženého.

7 Citovaná literatura

1. **Wooldridge, Michael.** *An Introduction to MultiAgent Systems.* Chichester (West Sussex) : John Wiley & Sons, 2002.

2. **Wooldridge, Michael a Jennings, Nicholas R.** Intelligent agents: Theory and practice. *Knowledge Engineering Review.* 1995, Sv. 10, 2, stránky 115-152.

3. **Newell, Allen a Simon, Herbert A.** Computer Science as Empirical Inquiry: Symbols and Search. *Communications of the ACM.* 1976, Sv. 19.

4. **Russel, Stuart.** *Rationality and Intelligence.* University of California : Computer Science Division, 2005.

5. **Klappenecker, Andreas.** *A First Look at Propositional Logic.* [Online] [Citace: 28. Listopad 2008.] http://faculty.cs.tamu.edu/klappi/cpsc289-f08/propositional_logic.pdf.

6. **Bratman, Michael E.** What is intention? *Intentions in Communication (P. R. Cohen, J. L. Morgan, and M. E. Pollack, eds.).* 1990.

7. **Dennett, Daniel C.** *The Intentional Stance.* Cambridge, Massachussets : MIT Press, 1987.

8. **Bratman, Michael E.** *Intention, Plans and Practical Reason.* Stanford, California : CSLI Publications, 1999.

9. **Kubík, Aleš.** *Inteligentní agenty: tvorba aplikačního software na bázi multiagentových systémů.* Brno : Computer Press, 2004.

10. **Wooldridge, Michael J.** *Reasoning about Rational Agents.* Cambridge, Massachusetts : MIT Press, 2000.

11. **Rao, Anand S. a George, Michael P.** BDI Agents: From Theory to Practice. *Proceedings of the 1st International Conference on Multi-Agent Systems (ICMAS-95).* 1995.

12. **Georgeff, Michael P. a Lansky, Amy L.** Reactive Reasoning and Planning. *In proceedings of 6th Conference on Artificial Intelligence (AAAI-87).* 1987.

13. **Zbořil, František.** *Agentní a multiagentní systémy: Studijní opora.* Brno : FIT VUT, 2006.

14. **Rao, Anand S.** AgentSpeak(L): BDI agents speak out in a logical computable language. *17th European Workshop on Modelling Autonomous Agents in a Multi-Agent World.* 1996.

15. **Brooks, Rodney A.** Intelligence Without Representation. *Artificial Intelligence.* 1991, Sv. 47.

16. **Dennet, Daniel C.** *Consciousness Explained.* New York : Hachette Book Group USA, 1991.

17. **Brooks, Rodney A.** How to Build Complete Creatures Rather than Isolated Cognitive Simulators. [autor knihy] Kurt Van Lehn. *Architectures for Intelligence.* Hillsdale, New Jersey : L. Erlbaum Associates, 1991.

18. **Brooks, Rodney A.** Intelligence Without Reason. *In the proceedings of 12th International Joint Conference on Artificial Intelligence (IJCAI).* 1991.

19. **Janoušek, Vladimír a Kironský, Elöd.** *Exploratory Modelling with SmallDEVS.* Brno : Vysoké Učení Technické, 2004.

20. **Zeigler, Bernard P.** *Object Oriented Simulation with Hierarchical, Modular Models.* New York : Academic Press, 1990.

21. **Mazal, Zdeněk.** *Modelování Uvažujících Agentů Petriho Sítěmi.* Brno : Vysoké Učení Technické, 2008.

22. **Mazal, Zdeněk, a další.** *PNagent: A Framework for Modelling BDI Agents using Object Oriented Petri Nets.* Brno : Vysoké Učení Technické, 2008.

23. **Češka, Milan, a další.** *Petriho sítě: studijní opora.* Brno : FIT VUT, 1996.

24. **Janoušek, Vladimír.** *Modelování Objektů Petriho Sítěmi.* Brno : Vysoké Učení Technické, 1998.

25. **Hu, Xiaolin a Zeigler, Bernard P.** Model Continuity to Support Software Development for Distributed Robotic Systems: A Team Formation Example. *Journal of Intelligent & Robotic Systems, Theory & Application, Special Issue: Multiple and Distributed Cooperating Robots.* 2004.

26. **Orság, Filip.** *Robotika: studijní opopa.* Brno : FIT VUT, 2006.

27. **FIPA.** *FIPA Agent Management Specification.* Geneva, Switzerland : Foundation for Intelligent Physical Agents, 2004.

28. **Squeak-SWiki.** Squeak Threading Model. *Squeak SWiki.* [Online] Squeak Smalltalk, 2007. Květen 31. [Citace: 21. Květen 2009.] http://wiki.squeak.org/squeak/382.

29. **Karlsson, Björn, a další.** Intelligent Sensor Networks - an Agent-Oriented Approach. *Workshop on Real-World Wireless Sensor Networks (REALWSN'05).* 2005.

Seznam použitých zkratek

ACL Agent Communication Language

BDI Belief Desire Intention

CTL Computational Tree Logic

CPN Colored Petri Nets

DEVS Discrete Event System Specification

FIPA Foundation for Intelligent Physical Agents

LORA Logic for Rational Agents

OOPN Object Oriented Petri Nets

PRS Procedural Reasoning System

UML Unified Modeling Language

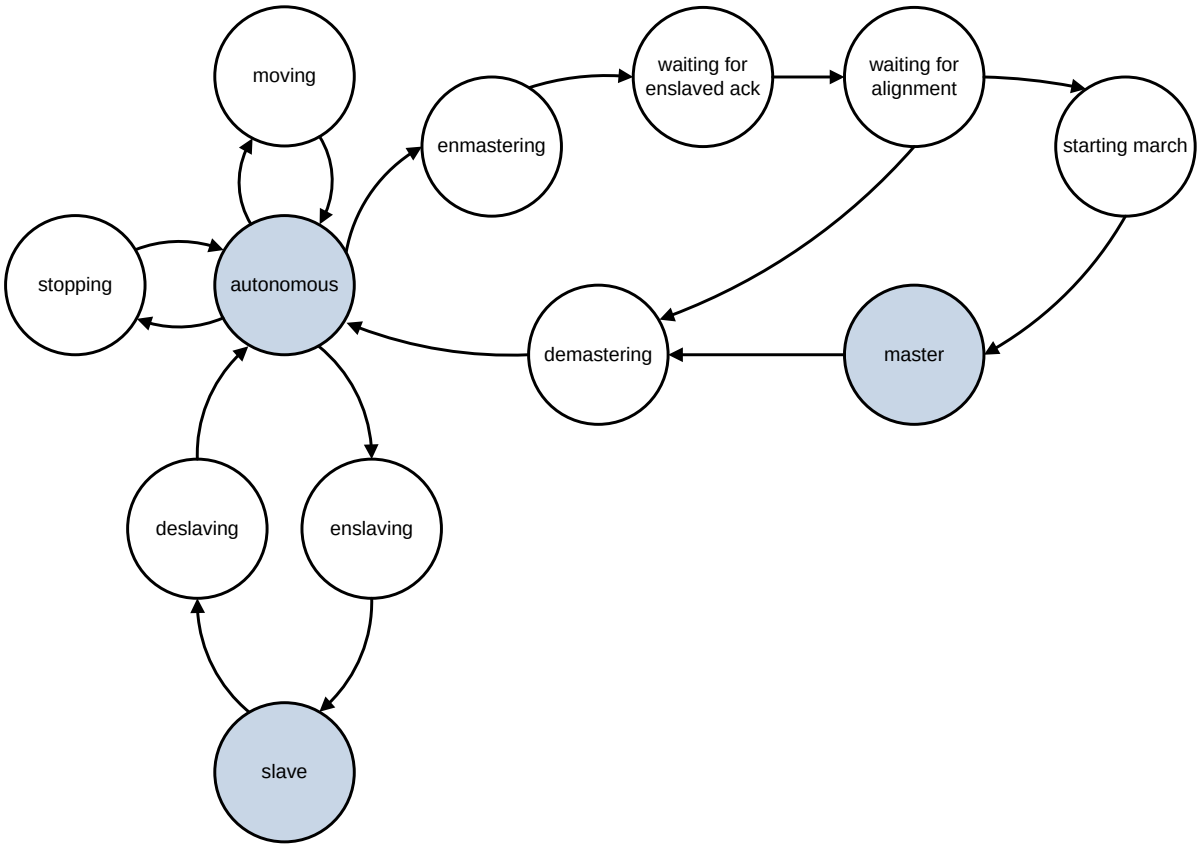
Přílohy

A Stavový diagram komponenty Message Handler v řešení Subsumpční architekturou.

B Stavový diagram komponenty Distance Handler v řešení Subsumpční architekturou.

Příloha A

Stavový diagram komponenty Message Handler



Příloha B

Stavový diagram komponenty Distance Handler