Aplikace multiagentního prostředí JADE

Application of multi-agent framework JADE

Bakalářská práce

Martin Štěpánek

Vedoucí bakalářské práce: Ing. Ladislav Beránek, CSc., MBA.

Jihočeská univerzita v Českých Budějovicích

Pedagogická fakulta

Katedra informatiky

2009

Prohlášení

Prohlašuji, že svoji bakalářskou práci jsem vypracoval/-a samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách.

V Českých Budějovicích dne

Anotace

Práce se zabývá realizací modelu multiagentního systému pro sledování a kontrolu výrobního či pracovního procesu. Systém je schopen reagovat na sledované jevy a provádět operace směřující k odstranění vzniklých problémů v procesu. Implementace je provedena pomocí Java Agent Development Framework (JADE) a jsou na ní demonstrovány vlastnosti a možnosti multiagentních systémů. Součástí práce je také základní seznámení s prostředím JADE.

Abstract

This work deals with realization of a model of multi-agent system for monitoring and supervising production or work process. System is capable of responding to monitored process events and taking actions leading to solving arised problems. Implementation is performed using Java Agent Development Framework (JADE) and there are demonstrated properties and capabilities of multiagent systems. This work also contains basic introduction of JADE framework.

Poděkování

Rad bych poděkoval Ing. Ladislavu Beranekovi, CSc., MBA.

Obsah

1 Úvod

S požadavky na řešení stále komplikovanějších úloh pomocí výpočetní techniky roste i složitost systémů, které toto řešení poskytují. Takový systém pak ztrácí schopnost přizpůsobit se novým podmínkám.

Zároveň s rozvojem síťových technologií, paralelizace a také rostoucí možnosti mobilních zařízení (především mobilních telefonů) vzniká tlak na využívání těchto technologií pro ještě větší integraci výpočetní techniky ve většině oblastí lidské činnosti. A právě budování složitých distribuovaných systémů je příležitost pro masové uplatnění multiagentních systémů.

Multiagentní systémy umožňují budovat komplexní systémy pomocí relativně jednoduchých částí, které je možné samostatně udržovat a vyvíjet. V minulosti však technologie s nimi spojená představovala spíš problém, kvůli nedostačujícím výkonům běžné výpočetní techniky. Jejich využití tak bylo omezeno na simulace a speciální úlohy.

Současný výkon výpočetní techniky tento problém odstranil a multiagentní systémy pronikají do dalších oblastí. Především v oblasti internetových aplikací je o tuto technologii zájem.

Java Agent DEvelopment Framework (JADE), na který je zaměřena tato práce, je jedním z existujících frameworků pro vytváření aplikací na bázi multiagentních systémů.

1.1 Cíle práce

Cílem této práce je seznámit čtenáře s koncepcí multiagentních systémů a možnostmi jejího využití pro vývoj aplikací. Práce poskytuje čtenáři základní vhled do teorie a její aplikace prostřednictvím JADE frameworku.

Jako prostředek k dosažení tohoto cíle byl zvolen praktický příklad, zabývající se možným využitím multiagentního systému pro monitorování podnikových procesů. Shanou je demonstrovat výhody agentově-orientovaného programování a potenciál v jeho praktickém uplatnění.

1.2 Struktura práce

Práce je členěna na 5 kapitol (včetně úvodu), jejichž obsah je následující:

Kapitola 2 seznamuje se základní teoretií, týkající se multiagentních systémů.

Kapitola 3 popisuje některé vybrané vlastnosti JADE frameworku, nezbytné pro pochopení jeho fungování.

Kapitola 4 se zabývá myšlenkou „paranoidního agenta“ a jeho použití pro monitorování podnikových procesů.

Kapitola 5 je závěrem, shrnujícím získané znalosti a hodnotící prostředí JADE na základě získaných zkušeností.

2 Problematika multiagentních systémů

2.1 Multiagentní systém

Multiagentí systémy (Multi-agent system, MAS) jsou jednou z oblastí zájmu distribuované umělé inteligence. Jsou to systémy složené z nezávislých (autonomních) jednotek interagujících v daném prostředí. Tyto jednotky nazýváme *agenty*.

Řešení problému pomocí multiagentního systému je založeno na rozdělení úlohy na množinu nezávislých podúloh a jejich řešení jednotlivými agenty (podobně jako u týmové spolupráce lidí).

Většina softwarových systémů se v průběhu svého vývoje dostává do situace, kdy je už není možné dále udržovat a vyvíjet jako celek kvůli rostoucí složitosti. Jejich rozdělení do nezávislých modulů je logickým vyústěním jejich dalšího vývoje. Cílem tohoto procesu je získat flexibilní systém, jehož jednotlivé části je možné vyvíjet odděleně a přitom zachovat komplexní funkčnost celku.

Mnoho současných softwarových systémů tak využívá některé prvky multiagentních systémů už nyní, přesto většinou ani zdaleka nevyužívají potenciál, který multiagentní systémy nabízejí.

2.2 Autonomní agent

Pojem agent je používán v mnoha různých oblastech a jeho konkrétní vlastnosti se liší. Myšlenka agentů vychází z pozorování chování živých organismů, zejména lidí. Obecně lze definovat agenta jako nezávislou entitu, která v daném prostředí vykonává akce, které směřují k dosažení určitého cíle.

Cíle agentů jsou závislé na konkrétním problému a. Díky rozdělení problému mezi jednotlivé agenty je možné využít výhody paralelního a distribuovaného prostředí a přitom díky nezávislosti agentů zachovat flexibilitu a snadnou modifikovatelnost systému.

Právě možnost jednotlivé agenty dále rozvíjet a libovolně je kombinovat v systému podle potřeby je jednou z jejich největších předností.

Další možné schopnosti agenta jsou dány konkrétním využitím, například schopnost komunikovat s ostatními agenty, uchovávat si model prostředí, uchovánat si vnitřní stav, ovlivňovat vlastní parametry (učení) a daší.

2.2.1 Chování agentů

Chování agenta v prostředí je většinou motivováno snahou o dosažení konkrétního cíle. Tato snaha může být explicitně definována stanovením modelu chování agenta (agent si cíl „neuvědomuje“) nebo se agent vlastními rozhodnutími přímo snaží tohoto cíle dosáhnout.

Chování agenta může být dvojího druhu:

reaktivní – agentovy akce jsou založeny na vnímání prostředí

proaktivní – agent je schopen provádět akce i bez podnětu z prostředí, tzn. je iniciativní.

Tyto dva druhy chování se mohou různým způsobem kombinovat a doplňovat.

Agenty lze podle principu rozhodování rozdělit na tři základní typy: reaktivní, deliberativní a sociální. Těmto třem typům agentů se budeme dále stručně věnovat.

2.2.2 Reaktivní agent

Reaktivní agent odpovídá na podněty z prostředí pomocí konkrétní množiny předdefinovaných reakcí. Rozhodování probíhá na základě porovnávání stavu vstupu s kritérii pro výběr reakce. Agent si neuchovává vnitřní reprezentaci prostředí.

Čistě reaktivní agent neobsahuje vnitřní stav, operuje přímo na principu zpětné vazby.

Pokud agent obsahuje vnitřní stav, lze jej chápat jako konečný automat. Takový agent je schopen složitějších reakcí, které však stále závisí jen na posloupnosti předešlých vstupů.

Možnosti reaktivního agenta jsou značně omezeny. Jedná se o nejjednodušší formu agenta. Přesto lze kombinací jednoduchých reakcí na různý vstup docílit poměrně komplikovaného chování.

2.2.3 Deliberativní agent

Deliberativní agent volí své akce na základě jeho snahy o dosažení určitého cíle. Agent rozhoduje na základě svých znalostí o prostředí a z dostupný akcí sestavuje, aktualizuje a optimalizuje plán, podle něhož dosáhne splnění cíle.

2.2.4 Sociální agent

Sociální agent je schopen volit své akce na základě předpokládaných reakcí ostatních agentů v systému. Agent při svém plánování nezahrnuje pouze aktuální stav prostředí a jednotlivých agentů, ale i jejich cíle, plány a případně modely jejich chování.

To umožňuje skupině agentů spolupracovat na splnění společného cíle, nebo snahu o vzájemnou spolupráci při plnění cílů jednotlivých agentů.

2.3 Prostředí a interakce agentů v MAS

Prostředí (nebo také okolí) je významnou součástí systému, dodávající konkrétní rámec možností. Agent vnímá prostředí pomocí omezené množiny vjemů a jeho schopnost reagovat je také omezena množinou proveditelných akcí.

V mnoha úlohách se jednotlivý agenti snaží sestavovat model prostředí. Ten umožňuje lepší plánování v komplikovaných prostředích (např. v úlohách robotiky).

V našem případě je prostředím distribuovaný výpočetní systém, nabízející agentům základní možnosti komunikace a mobility a prostředky jsou dostupné prostřednictví API programovacího jazyka.

Interakce agentů v tomto výpočetním prostředí má trojí povahu:

- **konkurence** – agenti vzájemně soutěží o omezené zdroje.
- **kooperace** – agenti vzájemně spolupracují při plnění svých individuálních cílů.
- **koordinace** – agenti vzájemně spolupracují na plnění společného cíle. Lze jí chápat jako speciální typ kooperace.

Velmi důležitým způsobem interakce je vzájemná komunikace agentů a vzhledem k jejímu významu v této práci se jí budeme dále věnovat.

2.4 Komunikace agentů

Jak již bylo uvedeno, velmi důležitým druhem interakce agentů je vzájemná komunikace.

Pomineme samotnou realizaci komunikace, která je závislá na konkrétním technickém řešení a soustředíme se především na její formu.

Komunikace agentů může probíhat dvěma základními způsoby:

1. Nepřímá komunikace – má povahu sdílené paměti, přes kterou probíhá výměna informací mezi agenty.
2. Přímá komunikace – realizována prostřednictvím cílené výměny zpráv.

Nepřímá komunikace je většinou realizována pomocí tzv. tabule. Jednotliví agenti na tuto tabuli zapisují znalosti a sestavují tak kolektivní množinu znalostí. Zároveň mohou číst a upravovat to, co bylo na tabuli zapsáno ostatními agenty.

Tabuli lze realizovat i v systémech, které jí přímo nepodporují. V tomto případě roli tabule zastává speciální agent, který slouží jako datové úložiště tabule a komunikuje s celou skupinou agentů pomocí adresného zasílání zpráv (případně pomocí broadcastingu).

Využití tabule je především v oblasti inteligentních agentů a řešení úloh pomocí znalostních systémů.

Přímá komunikace vyžaduje, aby byl každý agent v multiagentním systému jednoznačně identifikován. Zároveň je pro její fungování nezbytné, aby podoba zpráv byla standardizována. V neposlední řadě pak je třeba definovat pravidla, kterými se vzájemná komunikace agentů bude řídit.

Tyto požadavky řeší **jazyků komunikace agentů** (Agent Communication Language, ACL).

2.4.1 Jazyky ACL

Jazyky ACL představují základní rámec pro vzájemnou výměnu zpráv mezi agenty. Jejich hlavním smyslem je zajistit, aby agenti univerzálně rozuměli základnímu smyslu zprávy, i když nemusí nutně porozumět jejímu obsahu. Především jde o poskytnutí informací o zdroji zprávy, důvodu zprávy a formátu (jazyku) obsahu zprávy.

Jazyky ACL dále určují pravidla, kterými se komunikace agentů řídí. Poskytují standardní modely konverzace agentů tak, aby reakce agentů na konkrétní typy zpráv byly předvídatelné a v celkové komunikaci panoval řád.

Teoretickým základem těchto jazyků je **teorie řečového aktu** (speech act theory). Ta považuje řečovou komunikaci za akci, jejímž cílem je ovlivnit posluchače. Tuto akci lze vyjádřit prostřednictvím **performativů**.

Prostřednictvím performativů je možné kategorizovat řečové akty podle záměru jím vyjádřeného, např. dotaz – „ptám se“ nebo žádost – „požaduji“.

Řečový akt se rozděluje na tři složky: promluvový, nepromluvový a mimopromluvový akt. Promluvový akt představuje samotnou promluvu. Nepromluvový akt zařazuje promluvu do kategorie odpovídající akci řečníka. Mimopromluvový akt pak zahrnuje cíl promluvy ve smyslu zamýšleného ovlivnění posluchače.

2.4.2 FIPA-ACL

Jazyk FIPA-ACL je standart sdružení FIPA (Foundation for Intelligent Physical Agents – www.fipa.org), organizací zabývající se standardizací agentních technologií.

Jazyk definuje uzavřenou množinu dvaceti tzv. komunikačních aktů (performativy), kterých je možné využít k vzájemné komunikaci mezi agenty. Mezi nejpoužívanější patří například *inform*, *confirm*, *refuse* a *not-understood*.

Dále definuje protokoly, na jejichž základně lze vést složitější konverzace mezi agenty. Protokoly určují, jakými komunikačními akty mají jednotliví agenti v těchto konverzacích reagovat.

Co se samotného obsahu zprávy týče, FIPA-ACL neklade specifické požadavky.

Více o specifikacích FIPA-ACL lze nalézt v [11].

2.4.3 Obsah zprávy a ontologie

Při realizaci předávání dat mezi agenty je třeba zajistit, aby změna ve fungování jednoho agenta nevyžadovala změny v agentech s ním spolupracujících. Tím by byla narušena základní výhoda multiagentního systému.

Jazyky ACL řeší proces komunikace mezi agenty, ale samotný obsah zprávy závisí na konkrétní úloze a nelze jej všeobecně definovat.

Řešením tohoto problému představují **ontologie**. Jedná se o deklarativní popis konceptů (pojmů) a jejich vzájemných vztahů, které popisují určitou problematiku.

Pokud tedy dva agenti komunikují na bázi určité ontologie, lze jejich vnitřní implementaci měnit bez narušení této komunikace.

2.5 Praktické využití multiagentních systémů

Do nedávné doby bylo využití multiagentních systémů svázáno především se simulacemi ve vědeckém výzkumu nebo využití v robotice. To bylo zapříčiněno především nedostačujícím výkonem běžně dostupné výpočetní techniky.

V současnosti začíná mnoho společností jevit zájem. Multiagentní systémy pomalu pronikají do informačních systémů, internetových aplikací a dalších oblastí, souvisejících s počítačovými sítěmi.

3 JADE

3.1 Stručná charakteristika

JADE (Java Agent DEvelopment Framework) je softwarová platforma pro realizaci aplikací na bázi společenství softwarových agentů podle specifikací FIPA. Vznikl ve společnosti Telecom Italia v roce 1998 a je dále vyvýjen jako opensource (od roku 2000) pod licencí LGPL, umožňující její volné šíření a využívání jak v open source tak i v komerčních projektech.

Prostředí je založeno na programovacím jazyce Java a navrženo jako middleware vrstva mezi standartním JVM (Java Virtual Machine) a agenty, poskytující základní možnosti správy agentů, komunikace, mobility agentů, a především API urychlující vývoj agentově orientovaných aplikací.

Platforma je dostupná pod licencí LGPL, umožňující její volné šíření a využívání jak v open source tak i v komerčních projektech.

3.2 Popis architektury

JADE nabízí několik základních kamenů pro tvorbu a běh multiagentního systému.

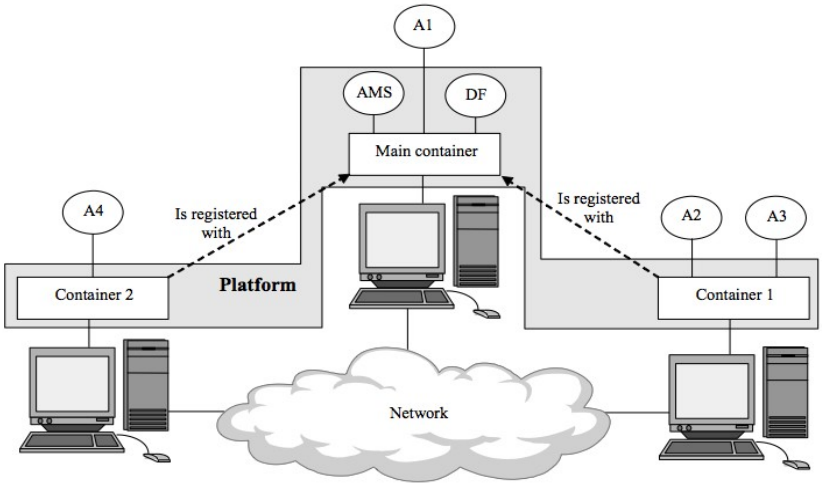
Jsou to:

- API, obsahující základní prvky pro tvorbu agentů.

- Běhové prostředí, které zprostředkovává správu životního cyklu agentů, vzájemnou komunikaci, mobilitu agentů v distribuovaném prostředí a některé další služby spojené s provozem a správou multiagentního systému.
- Základní ladící nástroje.

3.3 JADE Platforma

JADE je založeno na peer-to-peer agentní prostředí, složené z kontejnerů (Container) – spuštěných instancí runtime prostředí – obsahujících agenty. Tyto kontejnery tvoří společně platformu (Platform), která je spravována hlavním kontejnerem (Main Container).



Obr. 1: Schema JADE platformy

Každý agent je v rámci této platformy jednoznačně identifikovaný *identifikátorem agenta* (AID) a pokud zná jméno jiného agenta v rámci

platformy, může s ním komunikovat nezávisle na konkrétním umístění v rámci platformy. Komunikace probíhá vždy asynchronně.

3.3.1 Bližší pohled na platformu

Hlavní kontejner zajišťuje organizaci celé platformy. Kromě správy připojených kontejnerů obsahuje dva speciální agenty: *Agent Management System (AMS)*, *Directory Facilitator (DF*) a *Message Transport Service (MST)*. Celkový pohled na architekturu je zobrazen na obr. 1 (převzato z [8]).

AMS zajišťuje funkci bílých stránek, spravující seznam agentů a jejich AID v rámci platformy a jejich umístění v kontejnerech. Zároveň umožňuje v rámci platformy řídit životní cyklus agentů. Je unikátní pro konkrétní platformu.

DF zajišťuje funkci žlutých stránek, umožňujících vyhledávat agentům jiné agenty podle poskytovaných služeb.

MST realizuje předávání zpráv mezi agenty a odstiňuje agenty od konkrétní technické realizace komunikace. Díky tomu není nutné rozlišovat lokální komunikaci od komunikace prostřednictvím počítačové sítě.

3.4 Agent v JADE

JADE Agent je třída odvozená od třídy jade.core.Agent. Každý agent je v rámci svého kontejneru spuštěn jako samostatné vlákno, které cyklicky vykonává jednotlivá přidělená **chování** (behaviours).

Třída Agent je implementována velmi jednoduše. Zajišťuje základní implementaci komunikace agenta (správu fronty příchozích zpráv a funkci pro odesílání zpráv) a implementaci životního cyklu agenta. Dále pak nabízí základ podpory mobility agenta.

3.4.1 Životní cyklus agenta

Každý agent se může nacházet v jednom ze sedmi stavů, kompatibilních s FIPA Agent Management Specification. Tyto stavy jsou (převzato z [6]):

AP_INITIATED – *Inicializovaný*. Objekt agenta je vytvořen, nemá však zatím přiděleno AID a není zaregistrován v AMS a nemůže komunikovat.

AP_ACTIVE – *Aktivní*. Agent funguje normálně, přijímá zprávy a vykonává činnost. Jeho vlákno je aktivní.

AP_IDLE – *Neaktivní*. Agent nemá žádné aktivní chování. To může být způsobeno například tím, že jsou všechna blokována. V tom případě bude agent probuzen při příchodu zprávy.

AP_SUSPENDED – *Pozastavený*. Agentovo vlákno je pozastaveno a neprobíhá žádná činnost. Agent musí být aktivován externě, prostřednictvím AMS.

AP_WAITING – *Čekající*. Agent čeká na vnější podnět (většinou zprávu) pro pokračování v činnosti.

AP_DELETED – *Smazaný*. Agent již neexistuje v platformě, jeho vlákno je ukončeno.

AP_TRANSIT – *Na cestě*. Agent se nachází v zastaveném stavu a je přenášen do nového umístění. Týká se pouze mobilních agentů.

Změnu stavu agenta je možné vyvolat prostřednictvím AMS nebo agentem samotným.

3.4.2 Implementace činnosti – chování (Behaviour)

Pro implementaci činnosti agenta jsou použity třídy odvozené od třídy *Behaviour* (chování). Instance těchto tříd, přidané agentovi představují jednotlivé úkoly nebo činnosti, které má konkrétní agent vykonávat.

Základní implementace chování vyžaduje implementaci metody *action()* a metody *done()*.

Metody *action()* jednotlivých chování jsou spouštěny cyklicky na základě principu kooperativního multitaskingu. Po dokončení metody *action()* modelu chování je volána metoda *done()*. Pokud ta vrací true, je chování vyřazeno z dalšího cyklu.

Každé chování se může nacházet v jednom ze tří stavů:

- **aktivní** – jeho metoda action je právě prováděna
- **čekající** - všechny neblokované modely chování, kromě aktivního
- **blokované** – nebude prováděn, dokud agentovi nepřijde zpráva nebo neuplyne určitá doba, stanovená při jeho blokaci

Díky tomuto řešení je možné spouštět agenta jako jediné vlákno a odpadá problém se synchronizací přístupu ke sdíleným zdrojům. Dále je možné bezpečně zastavit běh agenta v okamžiku přepínání chování, což zjednodušuje implementaci mobility agenta.

Při implementaci chování agenta je třeba mít na paměti, že při spuštění konkrétního chování dojde ke spuštění jeho metody action a ta je prováděna do jejího ukončení. Pokud má chování vykonávat určitou činnost delší dobu nebo jí má vykonávat cyklicky, je třeba zajistit přerušování činnosti tak, aby bylo možné spouštět i další chování a především vestavěnou funkčnost agenta, která je obsluhována mezi spouštěním jednotlivých metod *action()*.

Důležité je zejména blokovat chování, která pro svou další činnost potřebují příchozí zprávu – metoda *block()*, nebo je jejich činnost možno přerušit na určitou dobu předáním metodě *block()* čas v milisekundách jako parametr.

Nakonec je možné spouštět chování v samostatných vláknech prostřednictvím třídy *ThreadedBehaviourFactory*. Zde je však potřeba

zajistit dodatečnou implementaci ukončování jednotlivých vláken a je třeba věnovat dodatečnou pozornost synchronizaci sdílených prostředků agenta.

3.4.3 Uchovávání vnitřního stavu agenta

Pro potřeby uchovávání vnitřních stavů agenta nabízí JADE několik jednoduchých možností.

V prvé řadě má každé chování možnost přístupu k instanční proměnné *myAgent*, která umožňuje přístup k agentovi, který toto chování spouští. Vzhledem k tomu, že celý agent běží v jediném vlákně, je možné přistupovat k instančním proměnným a metodám agenta bez potřeby řešit problém synchronizace.

Takto je možné v rámci agenta definovat potřebné vlastnosti a sdílet je mezi jednotlivými chováními.

Toto řešení je velmi jednoduché, jeho hlavní nedostatek je však v tom, že svazuje chování s konkrétní třídou agenta a brání tak v opětovné využitelnosti chování v jiném kontextu.

Druhá možnost myslí právě na tento problém. Každé chování obsahuje metody *getDataStorage()* a *setDataStorage()*. Pomocí těchto metod je možné nastavit chováním společné úložiště pro data v podobě mapy.

Jednotlivá chování pak získají sdílený objekt pomocí klíče, který mohou sdílet prostřednictvím jednoduchého rozhraní s definovanou řetězcovou konstantou, která tomuto klíči odpovídá.

Toto druhé řešení nabízí mnohem flexibilnější řešení vnitřního stavu agenta a lze ho považovat za preferované.

3.4.4 Základní chování

JADE poskytuje celou řadu připravených chování. Jedná se především o sadu základních typů z *jade.core.behaviours* a předpřipravené chování pro obsluhu některých komunikačních aktů.

OneShotBehaviour

OneShotBehaviour představuje jednorázovou akci. Jeho metoda *done()* vrací true a proto bude jeho metoda *action()* provedena pouze jednou.

CyclicBehaviour

CyclicBehaviour se provádí opakovaně, jeho metoda *done()* vrací vždy false.

Je vhodné pro implementaci cyklických činností, kdy jeho metoda *action()* představuje tělo cyklu.

CompositeBehaviour

CompositeBehaviour je základem pro chování složené z více jednodušších chování. Poskytuje základ pro správu těchto podřízených modelů. V rámci své metody *action()* spouští metodu *action()* jednoho podřízeného chování. Podřízené modely střídá na základě volání metod *startFirst()* a *startNext()*, jejichž implementace je věcí odvozených tříd.

Toto chování slouží jako základ pro dále uvedené *ParallelBehaviour, SequentialBehaviour a FMSBehaviour*.

ParallelBehaviour

ParallelBehaviour umožňuje vykonávat více podřízených modelů současně (střídají se cyklicky). Umožňuje dva módy dokončení – při dokončení jednoho podřízeného modelu nebo při dokončení všech.

SequentialBehaviour

SequentialBehaviour provádí sekvenci podřízených modelů, která se posouvá při dokončení aktivního. K dokončení chování dojde s dokončením posledního chování v sekvenci.

FMSBehaviour

FMSBehaviour řídí přepínání podřízených modelů chování na základě konečného stavového automatu.

Každý podřízený model chování definuje jeden stav automatu.
Při dokončení jeho činnosti je pomocí tranzitní tabulky a návratové

hodnoty modelu chování, získané z funkce *onEnd()* podřízeného modelu, vybrán další stav a spuštěn jemu příslušný podřízený model chování.

Celý model chování je dokončen při dokončení jednoho z konečných stavů (registrovaných pomocí metody *registerLastState()*).

3.4.5 Agent a GUI

Agent v JADE může mít i vlastní grafické uživatelské rozhraní, ať už pro monitorování jeho činnosti, nebo pro potřeby komunikace s uživatelem. Pro jeho vytvoření lze použít běžné nástroje, poskytované Javou.

Je však třeba zajistit synchronizaci mezi vláknem agenta a vláknem událostí GUI. To lze zajistit následujícím způsobem:

Akce přenášené z GUI na agenta lze provést přidáním odpovídajícího chování. Metody pro přidání chování jsou synchronizovány.

Agent pak může ovlivňovat GUI pomocí zařazování požadovaných operací pro vykonání vláknem událostí (u Swingu například metodou *SwingUtilities.invokeLater()*).

Přístup ke sdíleným datům je třeba ošetřit běžnými metodami synchronizace.

3.4.6 Komunikace

Agent může přijímat a odesílat ALC zprávy prostřednictvím metod *send()* a *receive()*. Zprávy jsou reprezentovány třídou *ACLMessage*, která v sobě implementuje všechny náležitosti, vyplývající z FIPA standardů.

Příchozí zprávy jsou řazeny do fronty zpráv, ze které je voláním metody *receive()* získat jednu zprávu. Zprávy zle získávat i selektivně pomocí šablon (třída *MessageTemplate*).

Při příjmu zpráv z více chování je třeba pamatovat na to, že příchod zpráv do fronty není synchronizován s přepínáním chování. Každé chování může zprávu vybrat hned po jejím příchodu a není předem určeno v jakém pořadí budou mít jednotlivá chování přístup k přijaté zprávě. Tento problém může být u složitějších agentů velmi nepříjemný.

3.5 Další vlastnosti JADE

JADE nabízí celou řadu dalších nástrojů a vlastností, jejichž popis však překračuje rámec této práce.

Podpora ontologií a jejich implementace prostřednictvím jednoduchých datových objektů je v praktické části této práci využívána, její popis lze nalézt v [9].

Další zajímavou vlastností JADE je podpora **mobilních agentů**, schopných přecházet mezi kontejnery nebo i mezi platformami. Agenti tohoto typu vyžadují speciální implementaci a jejich využití je velmi specifické. Podrobnější informace lze nalézt v [1] nebo [6].

3.6 Hodnocení JADE a srovnání ostatními prostředími

Jade implementuje multiagentní systém na velmi základní úrovni. Nabízí jen velmi málo pokročilých funkcí, napřed nenabízí podporu vyvozování a báze znalostí.

Je také problematické z hlediska ladění, protože poskytuje jen velmi omezené nástroje na odposlech komunikace agentů a odesílání zpráv. Jeho hlavní přednost zůstává především v jeho dostupnosti a možnosti modifikovat ho pro konkrétní potřebu.

Srovnáním JADE s ostatními multiagentními prostředími lze najít v práci Jana Šmajcla (viz. [3]). Jedná se o srovnání systémů pro modelování a JADE zde nemá příliš dobré hodnocení.

4 Monitorování podnikových procesů

Podnikový proces je posloupnost kroků vedoucích ke splnění určité úlohy v rámci firmy. Jednotlivé procesy mohou zahrnovat práci lidí i techniky jejich spolehlivé fungování je klíčové k úspěšnému podnikání.

V moderní firmě jsou tyto procesy do určité hloubky podchyceny výpočetními systémy. Systémy, které zajišťují například evidenci skladů, účetnictví, správu dokumentů a další úlohy, jsou běžnou součástí moderního podniku.

To přináší mnohé výhody, od prostého udržování záznamů a pořádku po možnost zpětně analyzovat a optimalizovat různé aspekty podnikání.

V praxi jsou však využívány softwarové produkty různých firem, které jsou mnohdy navrženy podle konkrétního modelu, který nemusí vyhovovat té které firmě. Pak dochází k různým problémům s přizpůsobení produktu potřebám firmy, nebo v horším případě snaha firmy přizpůsobit se produktu. Navíc je zde otázka spolupráce mezi různými systémy, která může působit značné komplikace.

Multiagentní systémy mohou v této oblasti poskytnout svou největší výhodu – flexibilitu. Mohou doplnit existující systémy ve slabých nebo hluchých místech a zajistit most pro spolupráci mezi různými systémy.

4.1 Úloha sledování podnikových procesů

Předpokládejme firmu, která se zabývá zakázkovou výrobou.

Mějme obecný výrobní proces, který se skládá ze čtyř hlavních kroků:

1. Vystavení výrobního příkazu - specifikuje co je potřeba vyrobit a v jakém množství.
2. Sestavení podkladů pro výrobu - řeší alokaci konkrétních výrobních prostředků a přípravu technologických podkladů.
3. Výroba - jedná se o sled úkonů, které jsou dány konkrétními parametry požadovaných výrobků. Z tohoto pohledu je tato část procesu unikátní pro každý druh výrobku a nemá praktický smysl jí přesněji definovat. Je plně v režiji odpovědných pracovníků.
4. Odvedení vyrobených dílů na sklad a potvrzení ukončení výrobního příkazu.

Základem pro krok 1 jsou data z informačního systému. Krok 2 může a nemusí vyžadovat lidskou spolupráci, krok 3 je na ní však založen.

V tomto případě vychází proces s informačního systému a jeho provádění je předáno člověku. Jeho zpětné napojení na informační systém je také závislé na člověku.

V ideálním případě by bylo nejlepší zajistit, aby veškeré potvrzovací úkony byly provedeny automaticky a tím zabráněno možným chybám. To však v mnoha případech není technicky proveditelné.

Proto je informační systém v situaci, kdy očekává určitou událost, ale nemá možnost žádným způsobem ovlivnit její vyvolání. Tato situace může vést ke zmatkům nebo problémům v navazujících nebo paralelně probíhajících úlohách.

Systém není proaktivní.

Zlepšení situace je možné dosáhnout následujícím způsobem:

Na základě standardních výrobních časů a termínů lze předpokládat určitý časový interval, ve kterém by ke konkrétní události mělo dojít. Pokud k události nedojde, je třeba "zjistit co se stalo".

Hodnotíme-li závažnost problému, který zapříčinil zpoždění, můžeme toto hodnocení založit pouze na jediné dynamické veličině - době zpoždění.

Nejjednodušší a v mnoha případech i nejefektivnější reakcí na tento problém je aktivní snaha systému kontaktovat odpovědné pracovníky.

Z této myšlenky vychází koncepce paranoidního agenta.

4.2 Paranoidní agent

Základní myšlenkou, skrývající se za paranoidním agentem je eskalace nátlaku na vyřešení problému. Agent, sledující vznikající problém postupně volí razantnější prostředky ve snaze vyřešit problém, až nakonec vyčerpá všechny možnosti.

Mějme agenta, jehož vnitřní stav se skládá z:

- úroveň nervozity *n,*
- krizový plán, který umožňuje agentovi provádět jednorázové akce na základě stupně nervozity, který je odvozen z prahových hodnot úrovně nervozity agenta.

4.2.1 Popis chování paranoidního agenta

Agent sleduje určitou veličinu, v našem případě čas uplynulý od převedení provádění výrobního procesu na člověka. Pokud tato veličina překročí hraniční mez, začne se zvyšovat nervozita agenta (předpokládejme lineární růst).

Pokud agentova nervozita dosáhne prahové hodnoty prvního stupně nervozity, agent reaguje na základě krizového plánu, definovaného pro tento první stupeň.

Pokud agent vyčerpá všechny možnosti krizového plánu a dosáhne maximální úrovně nervozity (1), nachází se ve stavu paniky a nemůže dál nic dělat se vzniklou situací. Stav paniky indikuje, že agent nedokáže řešit vzniklý problém a zároveň mu umožňuje tento problém indikovat a případně popsat dalším agentům, včetně kroků které podnikl a jejich úspěšnosti.

4.2.2 Krizový plán

Krizový plán představuje definici chování agenta pro jednotlivé stupně nervozity. Nejjednodušší formou takového plánu je prostý soupis po sobě jdoucích operací, které má agent provést (minimálně jedna).

Například:

n=>0.3 : kontaktuj pracovníka, pověřeného úkolem

n=>0.6 : kontaktuj všechny odpovědné osoby

n=>0.9 : kontaktuj jakéhokoli pracovníka, který může předat zprávu odpovědným osobám

a implicitně n = 1.0 : panika

Konkrétní podoba krizového plánu je závislá na úloze a možnostech agenta. Je například možné, že se agent na každém stupni snaží znovu provést nespěné kroky předešlých bodů plánu.

4.2.3 Akce

Agent nemusí mít žádné vlastní nástroje k nápravě problému. Tyto nástroje mu poskytují servisní agenti.

V takovém případě se agent snaží zajistit poskytnutí této akce prostřednictvím vyjednávání. Díky tomu může být akce v jednotlivých krocích velmi jednoduše definována, protože agent sám nemusí nutně „rozumět“ tomu, o co se snaží.

4.2.4 Implementace

Paraoidní agent je implementován třídou ParanoidAgent.

Agent využívá dva základní modely chování:

- *ConfirmationWaitingBehaviour* čeká na potvrzující zprávu o konkrétní události.
- *NervouslyWaitingBehaviour* které zajišťuje spouštění krizového plánu.

Akce prováděné krizovým plánem spočívají v zajištění předání zprávy odpovědným pracovníkům s popisem problému.

Seznam osob může narůstat s každým krokem plánu. Agent si udržuje seznam kontaktovaných osob a pokud dojde k vyřešení problému, oznámí to všem zůčastněným.

Při zajišťování služby předání zprávy agent postupuje tak, že tuto službu poptá u všech dostupných poskytovatelů. Po uplynutí minimální čekací doby vybere z došlých nabídek poskytnutí služby nejvýhodnější pro všechny osoby, které má kontaktovat.

Pokud pro některé osoby nabídka nepřišla, čeká dál. Také pokud pro některou z osob přijde nabídka výhodnější metody komunikace, využije i ji v případě, že nedošlo k potvrzení o přijetí zprávy předešlou metodou. Duplicitní předání zprávy v tomto případě není považována za chybu.

Při vyřešení problému pak agent odešle kontaktovaným osobám upozornění, že problém byl vyřešen.

Konkrétní fungování agenta je popsáno v přiložené dokumentaci.

4.3 Servisní agent

Servisní agent poskytuje konkrétní službu. Předpokládáme, že poskytovaná služba je omezená a nelze jí vyřizovat okamžitě. Agenti, kteří jí chtějí využít jsou proto řazeni do fronty požadavků a postupně obsluhováni.

Vyjednávání o poskytnutí služby probíhá na základě protokolu FIPA Contract Net Interaction Protocol. Jedná se o specifikaci dvoukolového vyjednávání v sekvenci požadavek – nabídka/odmítnutí – přijetí/odmítnutí – potvrzení o provedení/selhání. Podrobný popis viz. specifikace [11].

V našem případě se budeme věnovat skupině služeb, které souhrně označíme jako komunikace (communication). Jejich cílem je předat zprávu člověku pomocí různých komunikačních kanálů (např.: e-mail, sms, instant messaging, apod.).

Tyto služby můžeme rozdělit podle různých kritérií. Pro náš problém je nejpodstatnější rozlišení podle způsobu, jakým je možné monitorovat doručení zprávy.

Rozlišíme tři druhy komunikace:

- přímá – komunikace musí být přijmuta a je tím potvrzeno převzetí zprávy (např.: telefonní hovor, popup okno)
- nepřímá – doručení zprávy je potvrzeno, ale příjemce musí sám projevit zájem o její prijmutí (např.: sms, fax)
- bez potvrzení – doručení zprávy ani její převzetí není možné potvrdit, nebo je prodleva mezi odesláním a potvrzením doručení příliš dlouhá. (např.: e-mail, zvuková signalizace)

Druhým důležitým kritériem je dosah komunikace. Předpokládáme-li, že pracovník kterému je zpráva určena má na jejím základě provést nějaký úkon, musíme brát v úvahu, zda je vůbec v dostatečné fyzické blízkosti místa, kde tento úkon má provést.

Zde budeme rozlišovat pouze dvě možnosti:

- lokální – zpráva je doručena způsobem, u kterého lze předpokládat, že její adresát se nachází v místě, ze kterého může na zprávu reagovat (doručení zprávy na IM v rámci společnosti, doručení na notebook pracovníka v případě, že se jedná o zásah, který lze provést vzdáleně).
- obecná – nelze odhadnout, kde bude zpráva přijata (např.: sms mimo pracovní dobu)

Tyto základní parametry poslouží agentům k hodnocení výhodnosti nabídek jednotlivých poskytovatelů komunikace.

4.3.1 Agent osobní asisten – UserAssistantAgent

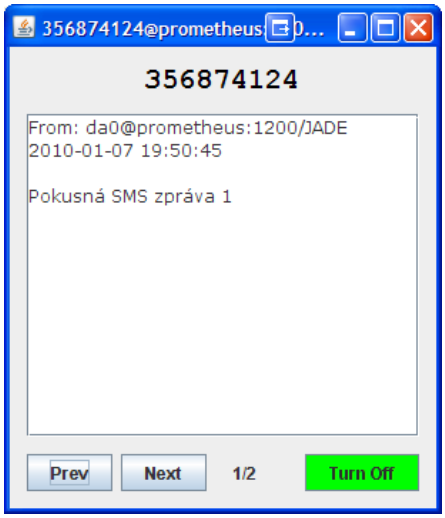
Agent osobní asistent (*UserAssistantAgent*) umožňuje předat jednomu konkrétnímu uživateli zprávu pomocí pop-up okna. Zprávy jsou předávány postupně a agent umožňuje maximálně 10 čekajících zpráv.

Implementace agenta demonstruje mimo jiné použití *ThreadedBehaviourFactory*.

Podrobnosti o jeho implementaci jsou popsány v přiložené dokumentaci.

4.3.2 SMSSendingAgent

SMSSendingAgent emuluje proces odesílání sms zpráv prostřednictvím SMS brány.



Obr. 2: PhoneEmulatorAgent

Agent umí rozdělovat dlouhé zprávy a zaznamenává si počty nepotvrzených zpráv pro jednotlivá telefonní čísla. Tak může omezovat odesílání zpráv na maximálně 10.

Implementace skutečného odesílání SMS zpráv je závislá na konkrétním zařízení a nebyla předmětem zájmu této práce. Místo toho jsou zprávy zasílány agentům *PhomeEmulatorAgent*, který zároveň demonstruje využití GUI v agentech (viz. obr. 2).

Popis implementace obou agentů obsahuje přiložená dokumantace.

4.4 Ukázkový model

Spuštěním agenta *DemoAdminAgent* lze spustit ukázkový model. Tento agent vytvoří několik *UserAssistantAgent* agentů a jednoho *SMSServiceAgent* agenta, některým uživatelům pak vytvoří emulovaný telefon, který bude náhodně vypínán a zapínán pomocí *DemoPhoneSwitchingAgent* agentů.

Model není plně funkční

5 Závěr

Problematika multiagentních systémů je velice rozsáhlá a není divu, že v ní panuje určitý chaos. Teoretie není jednotná a existuje mnoho odlišných trendů, které se vzájemně překrývají.

Jedná se však o perspektivní oblast, která zřejmě bude v budoucím vývoji výpočetní techniky zastávat významnou roli.

Tato práce se zaměřila na poměrně netradiční oblast využití multiagentních systémů v roli architektury softwarové aplikace. Toto zaměření bylo zvoleno ze dvou důvodů: práce zabývající se modelováním byla vypracována Janem Šmajclem v roce 2008 a prostředí JADE, na které se tato práce měla zaměřit, není vhodné pro tvorbu modelů.

V tomto směru je třeba přiznat, že nebylo dosaženo příliš uspokojivých výsledků.

Prostředí JADE, použité jako technologický základ, má mnoho nedostatků, které budou snad v budoucích verzích odstraněny. Jedním z nejzávažnějších je již dříve uvedené řešení příjmu zpráv agentem. Nedostatek ladících nástrojů je další nepříjemnou komplikací.

Jeho největší nedostatek je však nedostatečná dokumentace, především implementace komunikačních protokolů a ontologií.

Tyto problémy jsou hlavním důvodem nedostatků ve funkcionalitě praktické ukázky.

Tyto problémy jsou však řešitelné a vzhledem k otevřenosti platformy a šíření jejího využití lze očekávat další vývoj pozitivním směrem.

Reference

[1] BELLIFEMINE, Fabio, CAIRE, Giovanni, GREENWOOD, Dominic. *Developing Multi-Agent Systems with JADE*. Sussex, England : John Wiley & Sons Inc., 2007. 300 s. ISBN 978-0-470-05747-6.

[2] SPELL, B. *JAVA: Programujeme profesionálně*. Praha : Computer Press, 2002. 1022 s. ISBN 80-7226-667-5

[3] ŠMAJCL, Jan. Vývojová prostředí pro modelování multi-agentních systémů. [s.l.], 2008. 86 s. , 1 CD. Jihočeská univerzita v Českých Budějovicích. Pedagogická fakulta. Katedra informatiky. Vedoucí bakalářské práce Ing. Ladislav Beránek, CSc., MBA.

[4] MAŘÍK, V., ŠTĚPÁNKOVÁ, O., LAŽANSKÝ, J. a kol. *Umělá inteligence 2*. Praha : Academia, 1997. 373 s. ISBN 80-200-0504-8.

[5] MAŘÍK, V., ŠTĚPÁNKOVÁ, O., LAŽANSKÝ, J. a kol. *Umělá inteligence 3*. Praha : Academia, 2001. 328 s. ISBN 80-200-0472-6.

[6] BELLIFEMINE, Fabio, et al. *JADE Programmer's Guide* [online]. 2007-06-27 [cit. 2009-06-23]. Dostupný z WWW: <http://jade.tilab.com/doc/programmersguide.pdf>.

[7] BELLIFEMINE, Fabio, et al. *JADE Administrator's Guide* [online]. 2007-06-18 [cit. 2009-06-23]. Dostupný z WWW: <http://jade.tilab.com/doc/administratorsguide.pdf>.

[8] GIOBANNI, Caire. *JADE TUTORIAL : JADE programming for beginners* [online]. 2007-09-09 [cit. 2009-06-23]. Dostupný z WWW:

<http://jade.tilab.com/doc/tutorials/JADEProgramming-Tutorial-for-beginners.pdf>.

[9] GIOBANNI, Caire. *JADE TUTORIAL : Application-defined content languages and ontologies* [online]. 2002-06-30 [cit. 2009-09-11]. Dostupný z WWW: <http://jade.tilab.com/doc/tutorials/BeanOntologyTutorial.pdf>.

[10]Foundation for Intelligent Physical Agents. *FIPA Communicative Act Library Specification* [online]. 1996 , 2002/12/03 [cit. 2009-12-21]. Dostupný z WWW: <http://www.fipa.org/specs/fipa00037/SC00037J.html>.

[11]Foundation for Intelligent Physical Agents. *FIPA Contract Net Interaction Protocol Specification* [online]. 1996 , 2002/12/03 [cit. 2009-12-21]. Dostupný z WWW: <http://www.fipa.org/specs/fipa00029/SC00029H.html>.

[12] BURIAN, P. Servisně orientovaná architektura a Multiagentní systémy zvyšují Agilitu průmyslových podnikových ERP systémů [online]. Ústav počítačové a řídicí techniky, Fakulta chemicko-inženýrská, VŠCHT Praha , 2009 [cit 2008-09-11]. Dostupný z WWW: <http://si.vse.cz/archive/proceedings/2009/servisne-orientovana-architektura-a-multiagentni-systemy-zvysuji-agilitu-prumyslovych-podnikovych-erp-systemu.pdf>

Seznam ilustrací

Příloha A – Obsah doprovodného CD

1_ZADANI – zadání práce

2_TEXT – text práce ve formátu PDF

3_SOURCE – zdrojové kódy programu a dokumentace