Methods

In een eerdere editie van Java Magazine (nr. 3, juni 2009) werd veel aandacht besteed aan cloud computing, grid computing en genetische algoritmen. Exotische technieken met grote mogelijkheden, die de laatste jaren steeds meer volwassen zijn geworden. Deze technieken hebben gemeen dat ze nieuwe softwarearchitecturen en oplossingen mogelijk maken, evenals nieuwe bussiness-modellen. Een techniek die zeer goed in dit rijtje past is agent-technologie.

Agent-technologie

Van theorie naar praktijk met JADE

Agent-technologie is gebaseerd op het idee van zelfstandige software en samenwerking. De theorie hierachter ontstond halverwege de jaren '90. De explosieve groei van het internet zorgde voor een nieuwe visie op computertechnologie waarin interactie centraal stond. Via de digitale snelweg werden verspreide en heterogene computersystemen voor het eerst op grote schaal aan elkaar gekoppeld. En met de opkomst van het internet groeide het gedachtegoed over een nieuwe generatie softwarecomponenten. Software die aangepast was aan de nieuwe, dynamische omgeving van het WWW. Deze componenten moesten zichzelf kunnen redden en kunnen communiceren met andere componenten om samen te werken. De term softwareagent werd geïntroduceerd: een intelligente, sociale softwarecomponent met een grote mate van zelfstandigheid.

Een software-agent heeft een eigen agenda. Hij heeft een doel dat hij wil verwezenlijken en aannames over hoe hij dit wil doen. Daarnaast kan en wil een agent communiceren met andere agenten. Agenten communiceren met elkaar door berichten te versturen. Afhankelijk van de interne logica van de agent beslist deze hoe het bericht af te handelen. Met andere woorden, agenten communiceren met vragen, niet met opdrachten. In een agent-systeem werken agenten samen om problemen op te lossen. Ze leren van elkaar, delen kennis, en leveren gezamenlijk functionaliteit.

Voordelen

In een goed ontworpen agentsysteem hebben de agenten een 1 op 1 relatie met een fysiek of logisch onderdeel van de werkelijkheid. Neem bijvoorbeeld een agentsysteem voor routeplanning van een pakketservice. Een agent zou hierin een vrachtwagen kunnen representeren. Het doel van het agentsysteem zou kunnen zijn dat 'de vrachtwagens' onderling tot een optimale pakketverdeling en routeplanning komen.

Een agentsysteem heeft verschillende voordelen ten opzichte van conventionele applicaties. Het heeft geen 'single point of failure'. De agenten zijn immers verspreid en functioneren door communicatie. Wanneer een agent uitvalt kan de rest van het systeem blijven draaien of zelfs op de uitval reageren. Omgekeerd geldt dat eenvoudig nieuwe agenten kunnen worden toegevoegd aan het systeem. Een agentsysteem is robuust en zal ook bij onverwachte gebeurtenissen adequaat kunnen reageren. Ook als er veel onzekerheden in het spel zijn, zoals in het voorbeeld van de routeplanning, kan agenttechnologie krachtige oplossingen bieden. Tenslotte kan een agentsysteem als onafhankelijke laag toegevoegd worden aan (gedistribueerde) systemen, voor bijvoorbeeld optimalisatie of monitoring. Vooral in situaties waarbij functionaliteit geleverd wordt door verspreide systemen van diverse pluimage kan agenttechnologie een goede aanvulling zijn.

Er zijn verschillende agentplatformen beschikbaar waaronder JADE, Jack en Tryllian. In dit artikel gaan we verder in op JADE, een Open-Source agentplatform.

JADE

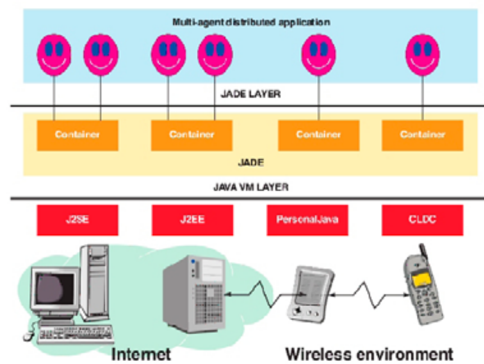
JADE staat voor Java Agent DEvelopment framework. JADE is een op Java gebaseerde middleware voor het ontwikkelen van agentsystemen. Het framework is ontwikkeld door Telecom Italia (sinds 1998) en wordt sinds 2000 door hen gedistribueerd onder de General Public License. JADE voldoet aan de standaarden van de Foundation for Intelligent Physical Agents (FIPA). Vandaag de dag is JADE een van de meest gebruikte agentplatformen.

Arjan Stoter
is Solution architect
bij Logica. Hij is bereikbaar
via arjan.stoter@logica.com.

Eduard Drenth
is Java Architect bij Logica.
Hij is te bereiken via email
eduard.drenth@logica.com.

Architectuur

Een JADE-platform bestaat uit 1 of meerdere agentcontainers (AC) en 1 of meerdere agenten. Een AC is een runtime-omgeving waarbinnen de agenten leven. De AC's van een JADE-platform kunnen verspreid zijn over verschillende soorten hardware in een netwerk of bestaan binnen één fysiek systeem. In een AC kunnen meerdere agenten leven. Figuur 1 geeft een schematische weergave van een JADE platform.



Figuur 1: schematische weergave van een JADE platform.

JADE is één van de meest gebruikte agentplatformen.

Naast AC's heeft ieder JADE-platform een zogenaamde main-container (MC). De MC is een unieke container. Deze wordt als eerste gelanceerd wanneer het platform wordt gestart en de overige containers registreren zich bij de MC. In de MC leven standaard drie speciale agenten: de directory facilitator (DF), het Agent Managment System (AMS) en de remote management agent (RMA).

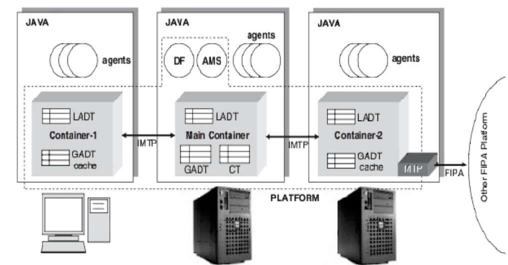
De DF is een yellow-pages service. De DF wordt gebruikt door agenten om hun service te registreren en om agenten te kunnen zoeken met een specifieke service. Met behulp van de DF/yellow-pages kunnen agenten bij elkaar gezocht worden op grond van een bepaalde service of vaardigheid. De DF illustreert daarmee een belangrijk kenmerk van een agentsysteem: functionaliteit in een agentsysteem wordt gerealiseerd door samenwerking op grond van vaardigheden van de verschillende agenten. Standaard leeft de DF in de MC maar deze kan ook in een andere container draaien.

De AMS is onder andere het registratiepunt voor de agenten. De identiteit van een agent wordt vastgelegd in een agent-identifier (AID) en geregistreerd in een global agent descriptor table (GADT), ook wel de white-pages. De AID bevat onder andere de naam van een agent en het adres. In tegenstelling tot de DF leeft de AMS altijd in de MC.

De derde agent in de MC is de RMA. Dit is een agent waarmee het platform beheerd en gemonitord kan worden. Hiervoor wordt standaard een GUI opgestart.

Tenslotte bevat de MC de container tabel (CT). Deze bevat de adressen en objectreferenties van alle containers in het platform. Samen leveren de DF, AMS en CT een topografie van het platform, zowel op locatie (adres en naam) als op functionaliteit (service).

Figuur 2 geeft een grafische weergave van de beschreven architectuur.



Figuur 2: de onderdelen van een JADE platform.

Hoewel de MC een unieke rol heeft binnen het platform vormt deze geen functionele bottleneck. De MC is in principe niet betrokken bij het verdere functioneren van het platform of de communicatie tussen agenten. Iedere container beschikt over een locale kopie van de GADT , de local agent descriptor table (LADT). Om te voorkomen dat de MC een 'single point of failure' vormt heeft JADE een 'main replication service'. Deze service bevat de 'main container replication service', waarmee meerdere kopieën van de MC gelanceerd kunnen worden. Eén van deze krijgt de rol van master toegewezen. De andere kopieën dienen als back-up. Wanneer een MC uitvalt kan het platform realtime overschakelen naar een van de back-up MC's.

Zoals eerder gezegd gebeurt communicatie tussen agenten met berichten. Deze berichten zijn opgemaakt in een specifieke taal. Deze taal, de agent communication language (ACL), voldoet aan de eerder genoemde FIPA-standaard.

Meest Belangrijke Klassen

De JADE API is zeer uitgebreid en intuïtief opgezet. De meest belangrijke klassen in JADE zijn Runtime, MainContainer, AgentContainer, Agent, ACLMessage en Behavior. De methoden binnen een klasse hebben over het algemeen intuïtieve namen.

Zoals in figuur 1 te zien is, omvat een JADE platform één of meer Java Virtual Machines (JVM's). Per JVM is er 1 instance van de Runtime klasse (het runtime-object is een singleton). Deze wordt onder andere gebruikt voor het aanmaken van containers en het lanceren van de MC.

De klasse MainContainer bevat de functies voor het beheer van de AC's, bijvoorbeeld voor creëren en registreren van AC's. Hierbij kan een AC op dezelfde machine draaien (local) of op een andere machine (remote). Bijvoorbeeld:

```
void addRemoteContainer(ContainerID cid)
void removeLocalContainer(ContainerID cid)
```

De klasse MainContainer is een extensie van de klasse AgentContainer. De klasse AgentContainer vertegenwoordigt de leefomgeving van één of meerdere agenten. Deze klasse bevat geen superspannende dingen behalve dat hij kan worden opgestart en zich kan aansluiten bij een bepaald platform. De AgentContainer bevat intuïtieve methoden om de toestand van een agent te beheren.
De klasse Agent is de basisklasse voor een agent. Deze klasse heeft een uitgebreide functionaliteit die we niet tot in detail gaan bespreken. Waar het op neer komt, is dat een agentobject methoden heeft om te kunnen reageren op de calls van de AgentContainer; met andere woorden, methoden om de toestand (lifecycle) van de agent te kunnen beheren. In zijn lifecycle kan een agent zich in vijf toestanden bevinden, waarbij transitie door de container geïnitieerd kan worden of door de agent zelf.

De standaard FIPA toestanden van een agent zijn:
- initiated: een agent is aangemaakt;
- active: een agent is actief, kan berichten verwerken;
- suspended: een agent is passief, kan geen berichten verwerken;
- waiting: een agent wacht (op resources), kan geen berichten verwerken;
- transit: een agent kan gemigreerd worden (naar een andere container), kan geen berichten verwerken.

Andere groepen van methoden binnen de klasse Agent hebben te maken met de communicatie en gedrag van een agent. Zoals bijvoorbeeld:

```
void postMessage(ACLMessage msg)
void send(ACLMessage msg)
void addBehaviour(Behaviour b)
void removeBehaviour(Behaviour b)
```

De voorbeelden 'postMessage' en 'send' maken gebruik van de klasse ACLMessage. ACL – zoals gezegd 'agent communication language' – maakt gebruik van standaardberichten die instanties zijn van de klasse ACLMessage. Deze klasse bevat methoden om een ACL-berichtobject te creëren en uit te lezen. Een dergelijk bericht bevat onder andere een lijst ontvangers, een afzender en een structuur (een ontologie). Methoden zijn bijvoorbeeld:

```
void addReceiver(AID r)
void getSender()
void setSender(AID s)
setOntology(java.lang.String str)
```

Naast functionaliteit voor lifecycle-beheer en messaging heeft een agent ook methoden om gedrag toe te voegen. Dit gedrag wordt vertegenwoordigd door de klasse Behavior. Dit is een abstracte klasse voor agentgedrag binnen JADE. Standaard heeft JADE een veeltal typen gedrag aan boord, vastgelegd in extensies van de klasse Behavior. Voorbeelden zijn CyclicBehavior (herhalend gedrag), OneShotBehavior (eenmalig gedrag) en TickerBhavior (periodiek gedrag).
De bovengenoemde klassen geven een aardig overzicht van het JADE-platform. Uiteraard bestaat het volledige platform uit aanzienlijk meer klassen en interfaces. Bovenstaande introductie zou je in staat moeten stellen om iets gemakkelijker je weg te kunnen vinden in deze API. De volledige JADE API is te vinden op http://jade.tilab.com/doc/api/.

Aan de slag

Theorie is leuk. Maar nog leuker is het om een JADE platform draaiende te zien. Hiervoor dien je eerst JADE te downloaden op http://jade.tilab.com/. De JADE core bestaat uit vier zip-bestanden, en is één keer te downloaden als jadeAll.zip. De 4 bestanden zijn:
- jadeBin.zip;
- jadeDoc.zip;
- jadeExamples.zip;
- jadeSrc.zip.

Pak deze bestanden uit. Na het uitpakken van alle zip-bestanden heb je een nieuwe map 'jade' met sub-mappen 'classes', 'demo', 'doc', 'lib' en 'src'. De JADE jar is te vinden in de map 'lib' .
We gaan eerst een platform starten. Open een console en ga naar de nieuwe map 'jade'. Typ het volgende om het platform te starten:

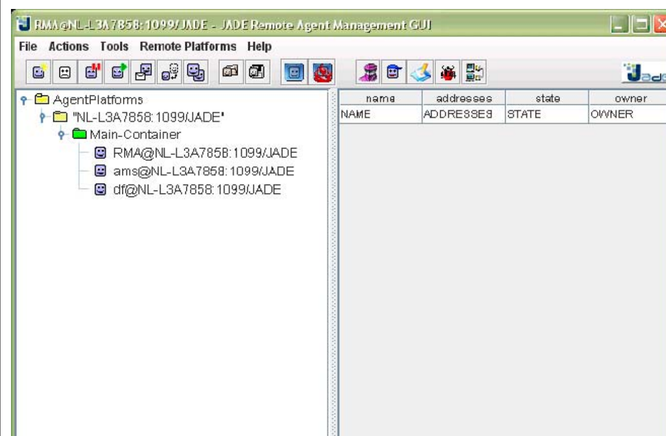
```
java -jar lib\jade.jar -gui
```

Er verschijnt nu een indrukwekkende lap tekst in de console, met termen die je misschien al bekent voorkomen uit het stukje over de JADE architectuur. Wat je zou moeten opvallen is de laatste melding:

```
Agent container Main Container@PLATFORM-NAAM is ready
```

We hebben eerder gezegd dat ieder JADE-platform een unieke agentcontainer bevat die de Main Container heet en die als eerste wordt gelanceerd. Dit is precies wat er is gebeurt. De optie –gui in je commando zorgt ervoor dat de RMA-GUI na een tijdje verschijnt (figuur 3).

Een agent-object heeft methoden om de agent te beheren.



Figuur 3: de RMA-GUI.

Het is leuk te zien dat een agent kan reizen tussen containers over een netwerk.

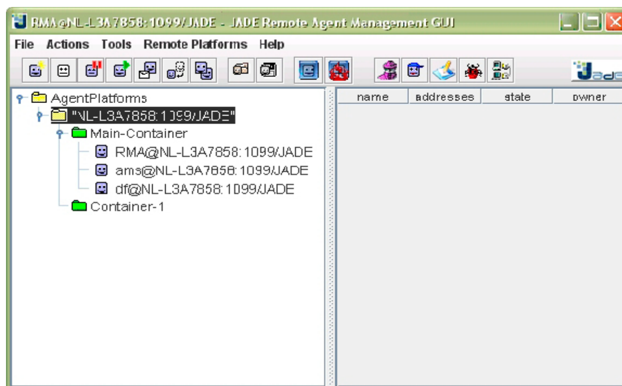
In het linkerveld van de RMA-GUI zie je een tree staan met als hoofdknoop ‘AgentPlatforms’. Wanneer je die uitklapt zie je de JADE-platformen die zijn opgestart. Als het goed is zie je een zogenaamde ‘agent platform description’ staan met je computernaam. Wanneer je deze uitklapt zie je alle containers die bij het platform horen, in dit geval 1 container: de Main-Container. Onder de knoop Main-Container zie je vervolgens alle agenten die draaien. Zoals beschreven in het stukje over architectuur zie je dat er drie agenten zijn gestart die draaien in de Main-Container: de RMA, AMS en DF.
We gaan nu een nieuwe container toevoegen aan het platform. Open een nieuwe console, ga naar de ‘jade’ folder en typ:

```
java -jar lib\jade.jar -container
```

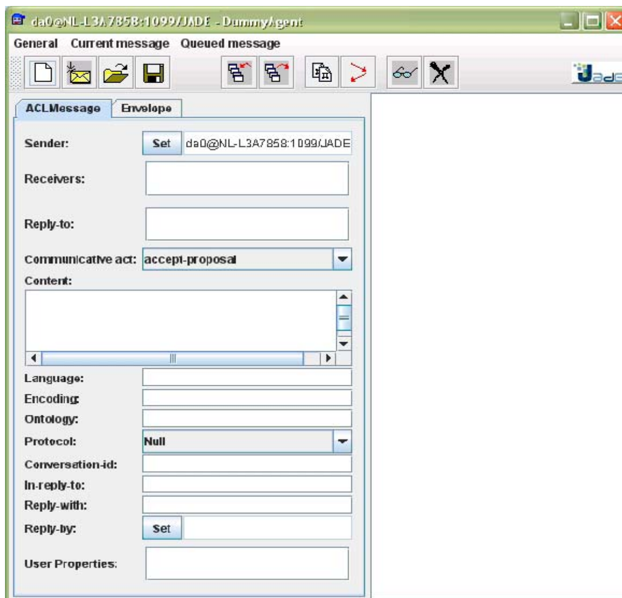
In je console zie je de melding:

```
Agent container Container1@PLATFORM-NAAM is ready
```

In de console waarin je JADE het eerst hebt opgestart is de melding verschenen:



Figuur 4: RMA-GUI met nieuwe container.



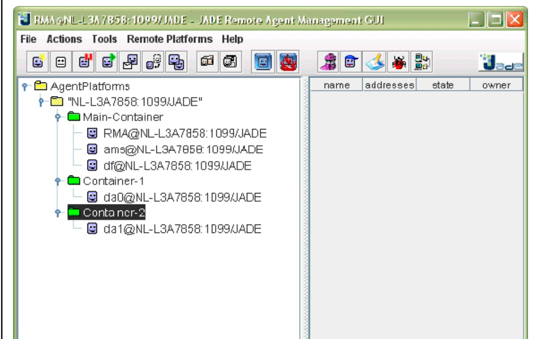
Figuur 5: controlescherm van een dummy-agent.

```
INFO:  --- Node<Container-1> ALIVE ---
```

Zie daar, er is een nieuwe container in het spel. Terug in de RMA-GUI zien we dat er inderdaad een nieuwe container is bijgekomen (figuur 4).

Herhaal bovenstaande stappen om een derde container op te starten.

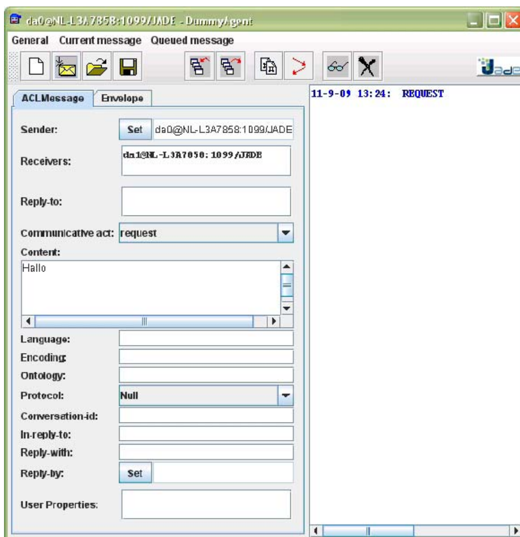
Nu we drie containers hebben, gaan we agenten aanmaken. Selecteer Container-1 en klik op het vierde icoontje van rechts. Hiermee lanceren we een zogenaamde dummy-agent. Een dummy-agent is een kant en klare agent waarmee je het platform kunt testen. We zien een scherm verschijnen (figuur 5). Dit is het controlescherm van de dummy-agent. Doe hetzelfde voor Container-2. Als het goed is ziet je RMA-GUI er nu ongeveerd uit als in figuur 6.



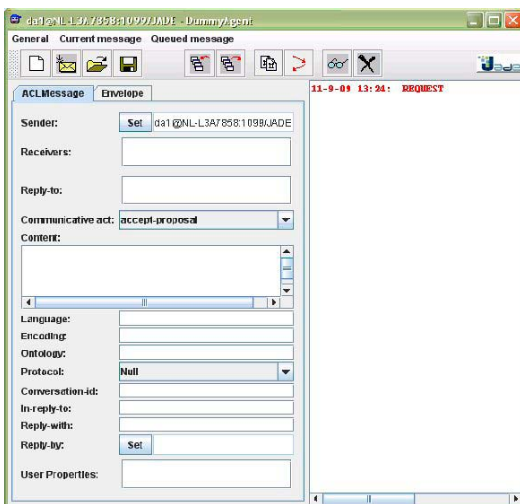
Figuur 6: RMA-GUI met 1 main-container, 2 containers en 2 dummy-agenten (da0, da1).

We gaan nu een testbericht versturen van dummy-agent 0 naar dummy-agent 1. Ga naar het controlescherm van dummy-agent 1. Selecteer de tekst in het veld ‘Sender’ en kopieer deze (ctrl-c). Ga naar het controlescherm van dummy-agent 0. Klik met de rechtermuistoets in het veld ‘Receivers’ en kies ‘add’. Plak (ctrl-v) de naam van dummy-agent 1 in het veld ‘Name’ en klik ‘OK’. (In het voorbeeld zou de naam zijn da1@NL-L3A7858:1099/JADE.) Selecteer bij ‘Communicative act’ de optie ‘Request’. Zet iets in het veld ‘Content’ en klik op het enveloppe- icoontje (‘Send the current ACL message’). In het controlescherm van dummy-agent 1 zien we dat er een request-bericht is aangekomen (figuur 7). We kunnen de inhoud van berichten bekijken door het bericht te selecteren en te klikken op het ‘bril-icoontje’. Dit laat de wijze zien waarop agenten communiceren.

Tenslotte is het nog leuk om te illustreren dat een agent werkelijk kan reizen tussen containers en -wanneer de containers zich op verschillende systemen bevinden – over een netwerk. In de RMA-GUI, selecteer dummy-agent 1 in Container-2 en gebruik de rechtermuistoets. Kies ‘Migrate Agent’ en vul in ‘Container-1’ (hoofdlettergevoelig). Klik



Figuur 7: verstuurd request-bericht, van dummy-agent 0 (boven) naar dummy-agent 1 (onder).



op 'OK'. En zie daar, dummy-agent 1 is verhuisd naar container 1.

Allemaal leuk en aardig, maar wat kun je er nu mee?

In een werkend agentsysteem lanceren we natuurlijk geen dummy-agenten maar werkelijke agenten die we zelf hebben gebouwd. Selecteer ter illustratie een container en klik op het meest linker icoontje. We kunnen dan een agent lanceren en een klasse kiezen voor deze agent. In de klassen van onze eigengemaakte agenten (extensies van klasse Agent) geven we logica mee over hoe ze moeten reageren op berichten van een bepaald type met een bepaalde inhoud. Door onderlinge communicatie worden dan de staten van agenten gewijzigd, berichten doorgestuurd, agenten verhuisd, aangemaakt en – niet geheel onbelangrijk - acties uitgevoerd. Dit is wat we bedoelen als we zeggen dat totale functionaliteit van het agentsysteem het gevolg is van onderlinge samenwerking en communicatie tussen zelfstandige, verspreide softwarecomponenten met ieder hun eigen logica.

In het bovenstaande hebben we het lanceren van het platform met de hand gedaan. In een werkend JADE-platform gebeurt dit met behulp van een configuratiebestand. Per Runtime (JVM) kan een configuratiebestand gebruikt worden. Het in de vingers krijgen van deze configuratie is een aardige klus en valt buiten deze kennismaking met JADE.

Wel willen we het nog even kort hebben over een aantal uitbreidingen voor JADE.

Uitbreidingen

Vanuit de JADE community worden interessante plugins en extenties onderhouden, bijvoorbeeld op het gebied van connectiviteit, encryptie, communicatiebeveiliging, XML en beheer.

Interessant om te noemen is LEAP. Met behulp van de LEAP add-on is het mogelijk om JADE-ondersteuning te bieden voor PDA's, mobiele telefoons en .NET. Een voordeel van LEAP is dat het om kan gaan met tijdelijke uitval van verbindingen en deze automatisch kan herstellen. Verder zijn op security-gebied uitbreidingen beschikbaar voor bijvoorbeeld JAAS en LDAP. (SSL, PKI en message encryptie zijn standaard aan boord.) En tot slot maakt een OSGI plugin geavanceerd laden van software en versiebeheer mogelijk.

Overwegingen

JADE is een volwassen framework voor het ontwikkelen van agentsystemen. Het heeft een uitgebreide en goed gedocumenteerde API waarmee je al snel aan de slag kunt. JADE is echter geen plug-and-play. Het kost tijd om het platform goed te leren kennen en je eigen agentapplicaties te ontwikkelen. Vooral in het begin krijg je veel informatie te verwerken. In veel gevallen heeft dit niet zozeer te maken met JADE zelf, maar de switch van conventionele softwarearchitecturen naar een agentstructuur.

De voordelen van agentsystemen zijn in dit artikel beschreven. Agentsystemen zijn robuust en hebben een grote mate van autonomie. Zij kunnen ons helpen in situaties die we moeilijk kunnen overzien, zoals planningsproblematiek.

Echter, de robuustheid van agentsystemen en het feit dat ze zich kunnen aanpassen aan een veranderlijke omgeving maakt dat hun gedrag moeilijker te voorspellen is. Vertrouwen in het systeem speelt daarom een grote rol. Hoogst waarschijnlijk zijn we allemaal bekend met fictie waarin autonome intelligente software de hoofdrol speelt. Het is misschien niet reëel om te denken dat een JADE-systeem de wereld gaat veroveren in de letterlijke zin van het woord. Maar het is belangrijk om te beseffen dat agentsystemen wel degelijk een onconventionele benadering van IT-oplossingen zijn, waarin we een stuk van de beslisvaardigheid en controle uit handen geven. «

Agentsystemen zijn robuust en hebben veel autonomie.