# PIONEER-ROBOTIN OHJAUS JADE-AGENTILLA

Antti Kukkasjärvi
Pro gradu -tutkielma
Tietojenkäsittelytiede
Itä-Suomen yliopisto
Tietojenkäsittelytieteen laitos
Tammikuu 2013

ITÄ-SUOMEN YLIOPISTO, luonnontieteiden ja metsätieteiden tiedekunta
Tietojenkäsittelytieteen koulutusohjelma

KUKKASJÄRVI, A.: Pioneer-robotin ohjaus JADE-agentilla
Pro gradu -tutkielma, 53 s.
Pro gradu -tutkielman ohjaajat: TkT Pekka Toivanen ja FT, DI Keijo Haataja
Tammikuu 2013

---

Avainsanat: FIPA, JADE, ohjelmistoagentti, Pioneer-robotti

JADE-agentit (Jade Agent Development Framework) ovat pieniä ohjelmia, jotka toimivat itsenäisesti ja viestittävät keskenään. Tässä tutkielmassa perehdytään Pioneer-robotin ohjaamiseen niiden avulla ja käydään läpi ohjelmistoagenttien arkkitehtuuria sekä eri protokollia.

Tutkielmassa perehdytään robotiikkaan yleisesti ja myös sen historiaan, nykypäivään sekä tulevaisuuteen. Lisäksi käydään läpi erilaisia robotteja ja niiden sovelluskohteita. Tarkemmin perehdytään Pioneer P3-DX -robotin ominaisuuksiin ja käyttökohteisiin sekä testataan kyseisen robotin toimintaa.

Tärkeimpänä osa-alueena tässä tutkielmassa on kuitenkin itse robotin ohjaus hyödyntäen JADE-ohjelmistoagenttia ja graafista ohjelmistoagenttien hallintatyökalua. Tutkielmassa esitellään malliesimerkkejä JADE-ohjelmistoagentista ja perehdytään tarkemmin ohjelmistoagenttien väliseen viestittelyyn sekä viestien hyödyntämiseen robotin liikkumisessa. Graafisesta JADE-ohjelmistoagenttien hallintatyökalusta käydään läpi keskeiset ominaisuudet kuvien avulla.

# Esipuhe

Tämä tutkielma on tehty Itä-Suomen yliopiston tietojenkäsittelytieteen laitokselle keväällä 2013. Tutkielman ohjaajana toimivat Pekka Toivanen ja Keijo Haataja, joille haluan osoittaa erityiskiitoksen.

Kuopiossa 9.1.2013

_____

Antti Kukkasjärvi

# Käsitteet ja lyhenteet

| | |
|---|---|
| ACL | Agent Communication Language |
| AMS | Agent Management System |
| API | Application Programming Interface |
| ARIA | Advanced Robot Interface for Applications |
| ASIMO | Advanced Step in Innovative Mobility |
| FIPA | Foundation for Intelligent Physical Agents |
| GPS | Global Positioning System |
| IEEE | Institute of Electrical and Electronics Engineers |
| JADE | Jade Agent Development Framework |
| XML | Extensible Markup Language |
| UML | Unified Modelling Language |

# Sisällysluettelo

# 1 JOHDANTO

Robotit ovat jo tärkeä osa nykyaikaa. Niiden tärkeimpänä tehtävänä on helpottaa ihmisen toimintaa. Ihmisen halu korvata hankalat ja työläät askareet robottien tehtäväksi on aiheuttanut robottien nopean kehittymisen. Robotit pystyvätkin tekemään monia asioita huomattavasti tarkemmin ja nopeammin kuin ihminen. Joitain asioita ei ole edes mahdollista toteuttaa ilman robottien apua. Hyvänä esimerkkinä on teknologiateollisuus, joka on nykyään niin pikkutarkkaa työtä, ettei siihen ihminen yksin pysty.

Tämän tutkielman sisältö koostuu useasta eri osa-alueesta. Tarkoituksena on perehtyä robotiikkaan yleisesti, muun muassa sen historiaan, laitteisiin ja tulevaisuuteen. Lisäksi syvennetään tietämystä ohjelmistoesimerkillä, jolla ohjataan Pioneer P3-DX -robottia. Kyseinen robotti on pyörillä liikkuva ja suunniteltu opetus- sekä tutkimuskäyttöön. Sen tärkein tehtävä on liikkuminen ja siihen on mahdollista kehittää erilaisia kohdetunnistimia hyödyntäviä sovelluksia. Tässä tutkielmassa on perehdytty myös hieman robottien kohteiden tunnistamiseen ja liikeratojen määrittämiseen.

Pioneer P3-DX -robotin ohjaamiseen käytetään tässä tutkielmassa JADE-ohjelmistoagenttia (Jade Agent Development Framework). Se on täysin itsenäinen ohjelma, jolle voi lähettää viestejä toisella ohjelmistoagentilla. Kaikessa yksinkertaisuudessaan tietokoneelta lähetetään Pioneer-ohjelmistoagentille viestejä ja ohjelmistoagenttien tehtävänä on kommunikoida Pioneer-robotin oman ARIA-ohjelmistorajapinnan (Advanced Robot Interface for Applications) kanssa.

Tutkielmassa perehdytään myös tarkemmin JADE-ohjelmistoagenttiarkkitehtuuriin ja sen eri protokolliin (luku 2). Ohjelmistoagentteja ohjataan niiden omalla graafisella JADE-kehyksellä, jonka toimintaan perehdytään selvin esimerkein. Samaa kehystä käytetään myös Pioneer-robottia ohjaavan Pioneer-ohjelmistoagentin ohjauksessa (luku 3). Kehyksestä saa helposti tietoa ohjelmistoagenttien välisestä viestiliikenteestä ja viestien sisällöistä.

Pioneer-robottia testataan mahdollisimman monipuolisesti sekä sitä ohjaavan ohjelmistoagenttiteknologian että sen liikkumisen osalta (luku 4). Testien pohjalta havainnoidaan robotin liikkeitä koordinaatiston avulla. Lopuksi luodaan yhteenveto tutkielman sisällöstä ja pohditaan saavutettuja tuloksia (luku 5).

# 2 JADE

JADE on täysin Java-kielellä toteutettu ohjelmistoagenttien kehitysympäristö. Agenttien standardoinnista vastaa FIPA-standardointijärjestö (Foundation for Intelligent Physical Agents). Agentit ovat ohjelmiston paloja, jotka kommunikoivat keskenään joko itsenäisesti tai ohjelmiston käyttäjän komentamana.

Luvussa 2.1 perehdytään tarkemmin FIPA:an, joka on ohjelmistoagenttien standardointiin perehtynyt organisaatio. Luvussa kerrotaan myös tarkemmin, mitä ohjelmistoagentilla tarkoitetaan ja minkälainen on sen elinkaari. Lisäksi luvussa käydään läpi kahta FIPA:n määrittelemää protokollaa. Ohjelmistoagenttien arkkitehtuuria ja luokkarakennetta käsitellään luvussa 2.2. Luvussa 2.3. kerrotaan JADE-kehitysympäristön eri agenteista ja siitä, miten agentteja hallitaan graafisessa JADE-ikkunassa. Luku 2.4 perehtyy agentin sisäisen rakenteen selvittämiseen ja luku 2.5 sisältää yhteenvedon luvun 2 aliluvuista.

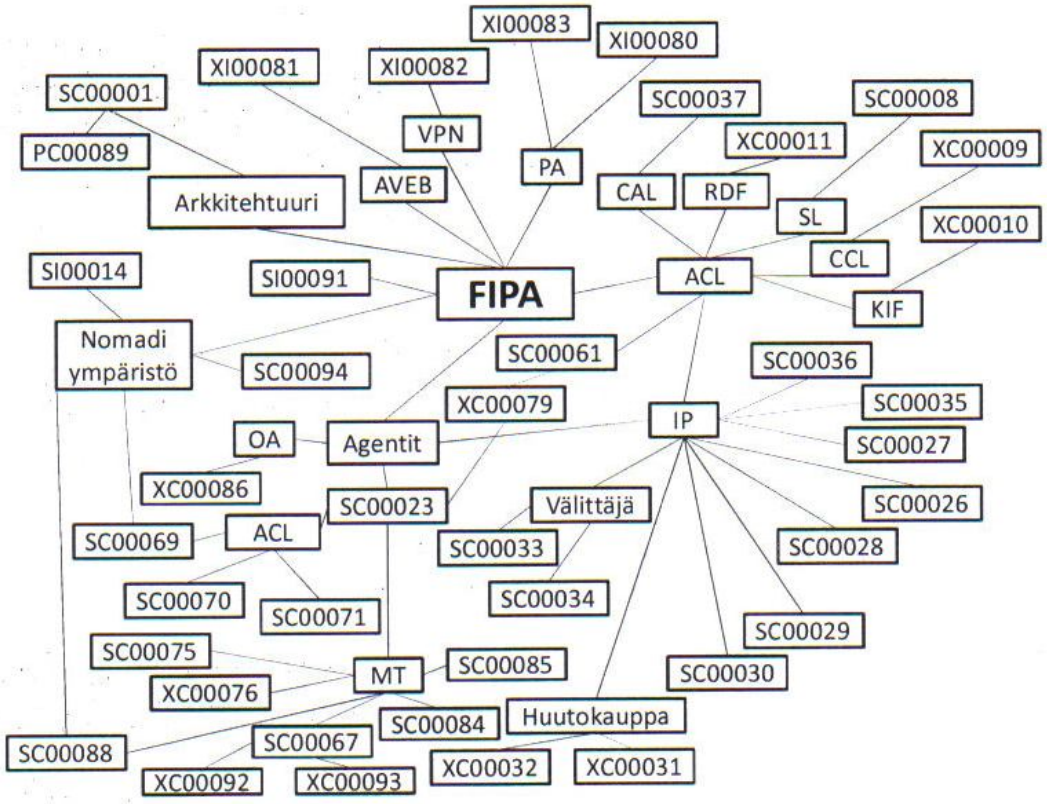
## 2.1 FIPA

FIPA on IEEE-standardointijärjestöön kuuluva organisaatio, joka aloitti toimintansa vuonna 1996. Sen tehtävänä on tuottaa agenttipohjaiseen teknologiaan perustuvia standardeja. FIPA:lla on ollut tärkeä rooli erilaisten agenttijärjestelmien kehityksessä ja se hyväksyttiin IEEE-standardien joukkoon vuonna 2005. [FIP13]

FIPA:n perusajatuksen viisi päämäärää ovat: [BCG07]

1. Vanhojen ja uusien ongelmien ratkaiseminen agenttiteknologian avulla.
2. Jotkin agenttiteknologiat ovat saavuttaneet merkittävän kypsyystason.
3. Standardointi mahdollistaa agenttiteknologioiden hyödyllisen käytön.
4. Positiivisia tuloksia on saatu geneeristen teknologioiden standardoinnilla.
5. Kielen ja infrastruktuurin standardointi mahdollistaa avoimen yhteistoiminnan ja agenttien sisäisillä mekanismeilla ei ole tärkeää merkitystä.

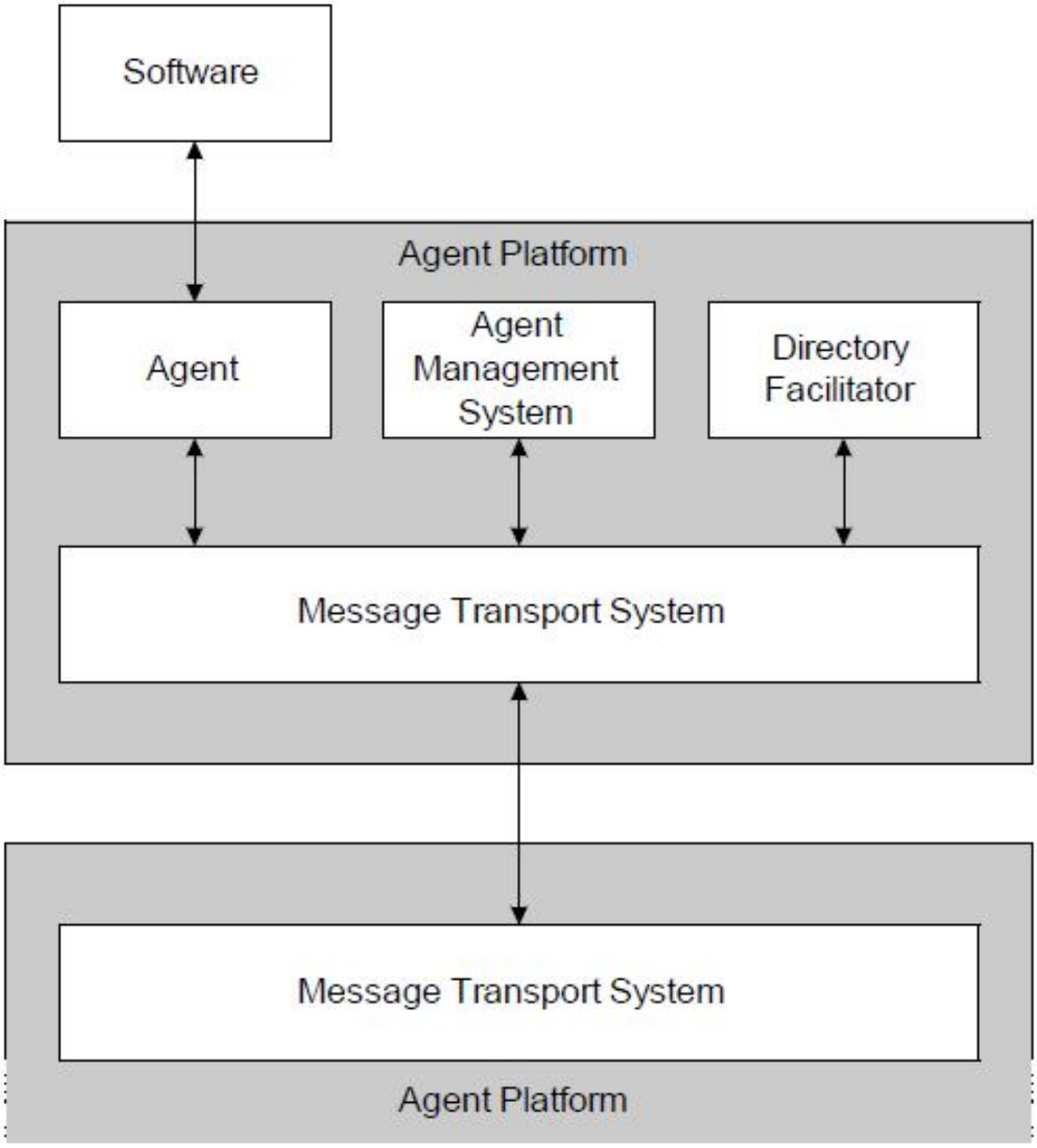
Standardeja on vuosien saatossa syntynyt useita ja niistä uusimmat perustuvat FIPA 2000 -standardiryhmään. Joanna Niininen [Nii09] selvitti omassa gradussaan standardien suhteita, jotka selviävät hyvin kuvasta 1. Suurin osa standardeista on kehitetty agenttien väliseen kommunikointiin.




**Kuva 1. FIPA 2000 -spesifikaatiot. [Nii09]**

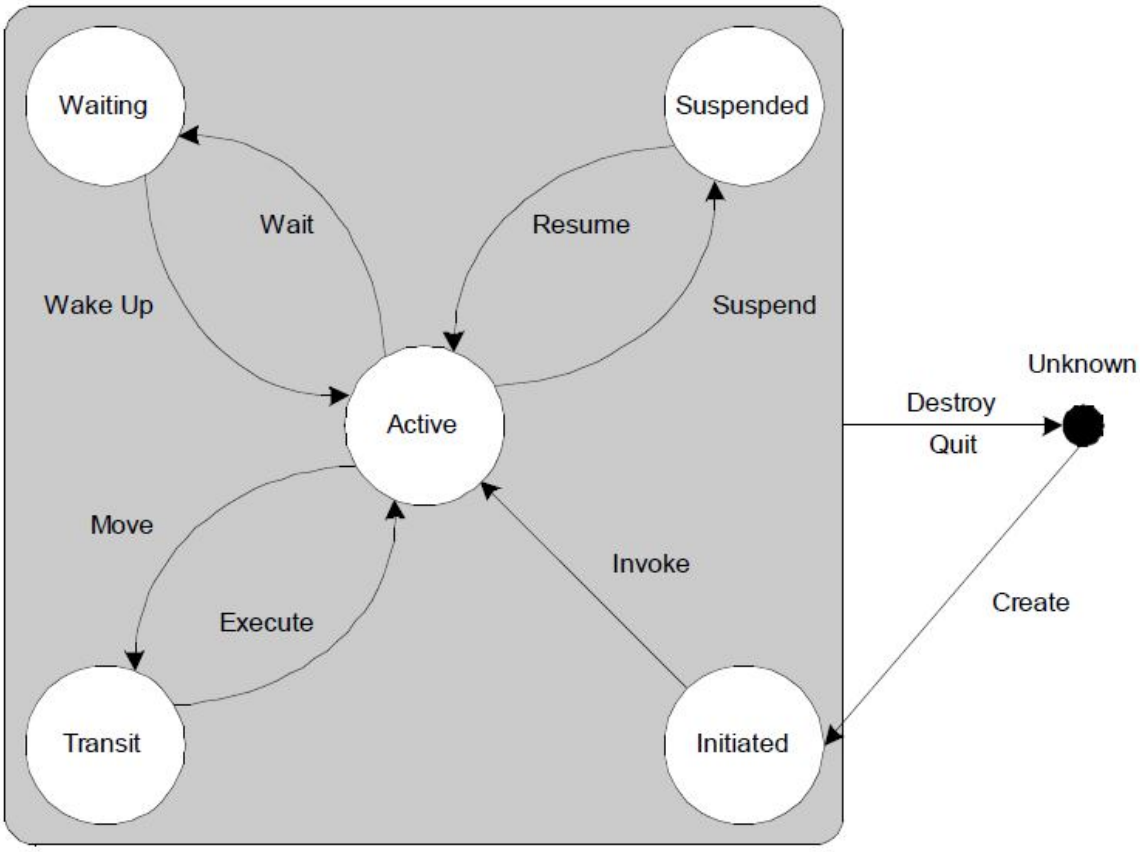
Ohjelmistoagentti on pala ohjelmistoa, joka kommunikoi joko toisten agenttien tai ohjelmiston käyttäjän kanssa. Agentin toiminta on itsenäistä, koska se pystyy toimimaan ilman suoraa komentoa ihmisiltä tai muilta ohjelmiston agenteilta ja se osaa myös kontrolloida omia toimintojaan sekä tilaansa. Agentti käyttäytyy kuitenkin myös sosiaalisesti, koska se kykenee toimimaan tietyssä järjestyksessä muiden agenttien tai ihmisten kanssa saavuttaakseen päämääränsä. Lisäksi se havainnoi omaa ympäristöään ja vastaa ajoittaisilla muutoksilla, joten se on myös reagoiva. Ohjelmistoagentti kykenee ennakoimaan ja sen onkin mahdollista saavuttaa päämäärätietoinen käytös tekemällä aloitteita. Agentit voivat olla myös siirrettäviä eli ne voivat toimia erilaisissa verkoissa. [BCG07]

FIPA:n agentit esitetään kuvan 2 kaaviossa. Ohjelma kommunikoi agentin kanssa, joka kuuluu agenttialustaan (Agent Platform). Agentti välittää viestejä toisille agenttialustoille viestien kuljetuspalvelua (Message Transport System) hyödyntäen. Agenttialustan hallintajärjestelmä (Agent Management System) hallinnoi agenttialustaa. Se päästää agenttialustalle vain ne viestit, joilla on pääsyoikeus sinne. Jokaisen agentin, joka haluaa kommunikoida agenttialustan kanssa, pitää rekisteröityä agenttialustan hallintajärjestelmään. Agenttialusta voi sisältää lisäksi palveluhakemiston (Direcrory Facilitator), jossa ylläpidetään listaa palveluista, joita tarjotaan toisille ohjelmistoagenteille. [Fou04]




**Kuva 2. Agenttien hallinta. [Fou04]**

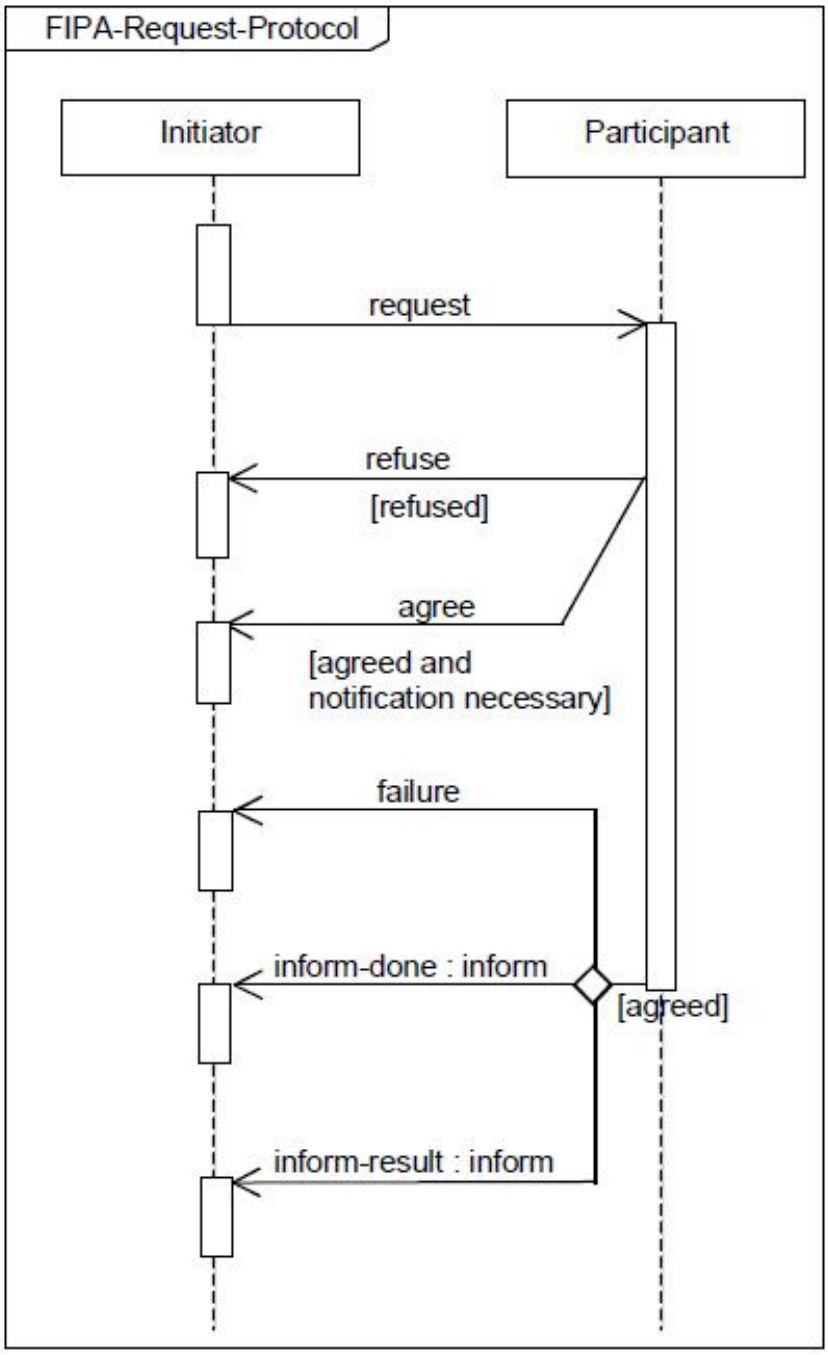
Ohjelmistoagentin elämänkaari on piirretty kuvaan 3. Agentilla on olemassa erilaisia tiloja, joita ovat aktiivinen (Active), alustettu (Initiated), odottava (Waiting), keskeytetty (Suspended) ja transitio (Transit). Aktiivinen tila on käytössä silloin, kun agentti on käynnistetty ja viestien välitys sujuu normaalisti. Alustetussa tilassa agenttia ei ole vielä käytetty ja se odottaa tulevaa ”Create”-käskyä. Odottavassa tilassa agentti on laitettu ”odotus”-tilaan ja se siirtyy ”Active”-tilaan ”Wake Up”-käskyllä. ”Keskeytys”-tilassa agentin suoritus on jostain syystä keskeytetty. Agentti odottaa, että toiminta käynnistetään uudelleen ”Resume”-käskyllä. Alustetussa, odottavassa ja keskeytetyssä tilassa agentti ei lähetä eikä vastaanota viestejä. Viestien puskurointi näissä tiloissa on mahdollista kunnes agentti palaa aktiiviseen tilaan. ”Transit”-tila on tarkoitettu liikkuville agenteille, jolloin viestit puskuroidaan tai lähetetään agentin uuteen kohteeseen. Agentin toiminta lopetetaan ”Quit”- tai ”Destroy”-käskyillä. Agentti voi olla huomioimatta ”Quit”-käskyä, mutta ”Destroy”-käsky lopettaa aina agentin toiminnan. [Fou04]




**Kuva 3. Agentin elinkaari. [Fou04]**

FIPA on kehittänyt standardeja agenttien väliseen kommunikointiin. Standardeilla varmistetaan se, että saadaan yksinkertaistettua agenttien välistä kommunikointia. Uusien agenttien luominen on helpompaa silloin, kun noudatetaan tiettyjä standardeita. FIPA pitää kirjaa eri kommunikointiprotokollista sen julkisessa kirjastossa (FIPA Communicative Act Library). [Fou04]

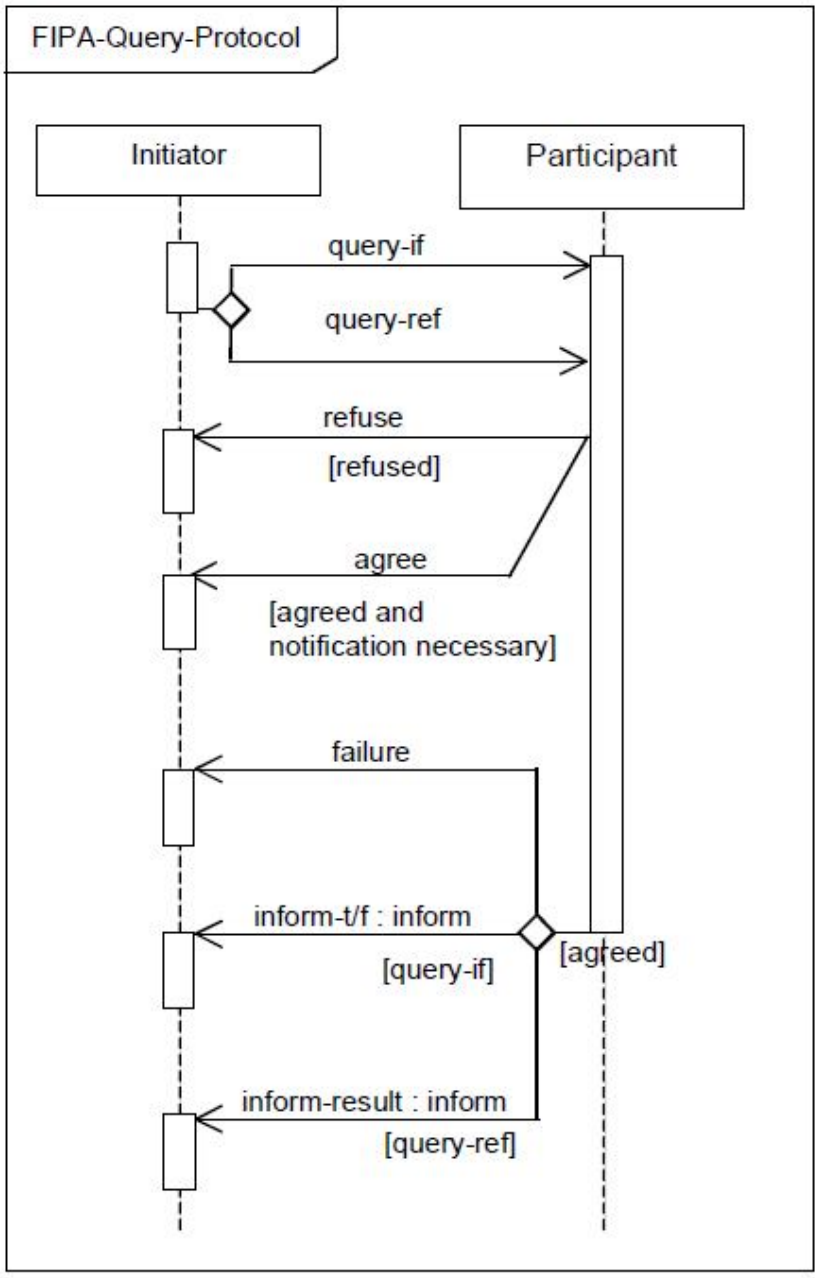
Kuvassa 4 on FIPA:n määrittelyjen mukainen pyyntöprotokolla (Request-Protocol).




**Kuva 4. FIPA:n määrittelyn mukainen pyyntöprotokolla. [Fou02]**

Lähettävä agentti (Initiator) lähettää pyynnön vastaanottavalle agentille (Participant). Vastaanottaja voi hyväksyä (Agree) tai kieltäytyä (Refuse) pyynnöstä. Mikäli vastaanottaja hyväksyy pyynnön, suoritetaan lähettäjän määräämä tehtävä. Vastaanottaja suorittaa tehtävän ja vastaa lähettäjälle, onnistuiko se. Mikäli tehtävä onnistuu, lähetetään joko ”Inform-done”- tai ”Inform-result”-viesti. Inform-done on ilmoitus tehtävän onnistumisesta ja ”Inform-result”-viesti sisältää lisäksi myös tietoa tehtävän suorituksen tuloksista. Tehtävän epäonnistumisesta ilmoitetaan ”Failure”-viestillä. [Fou02]

Kuvassa 5 on FIPA:n määrittelyn mukainen tiedusteluprotokolla (Query-protocol).

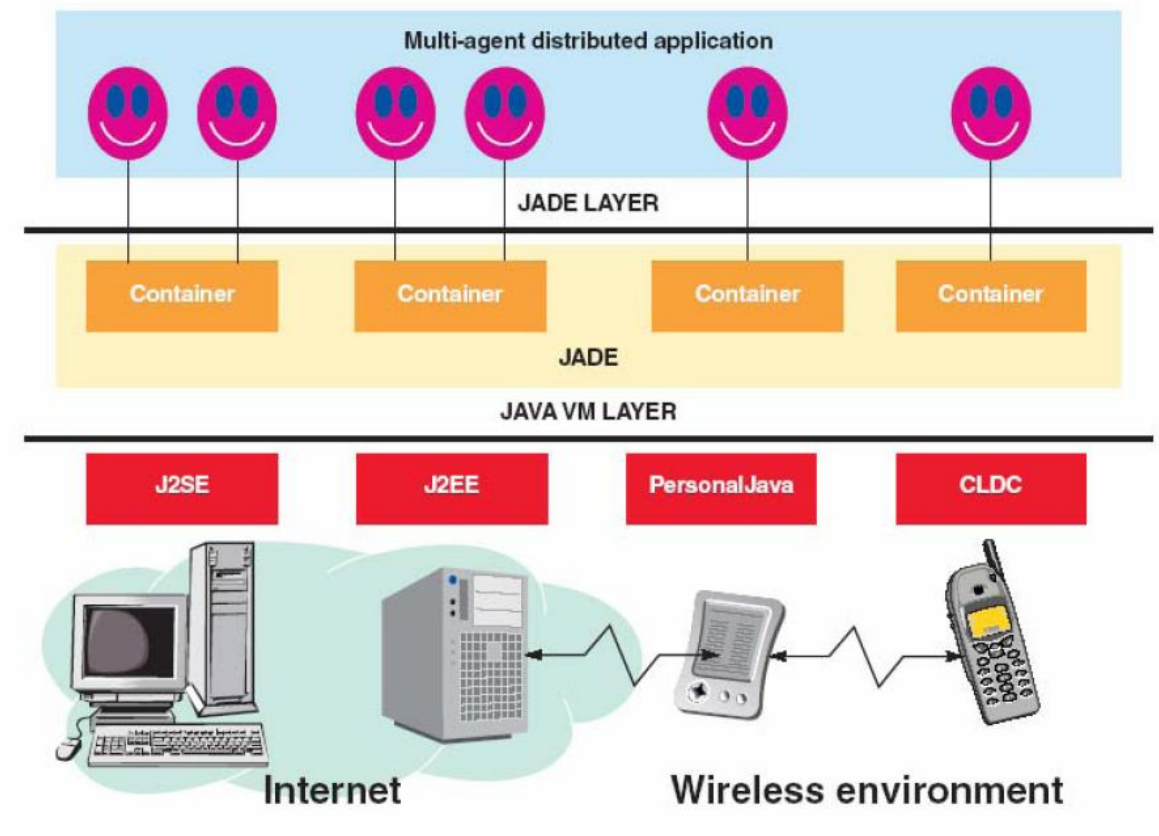



**Kuva 5. FIPA:n määrittelyn mukainen tiedusteluprotokolla. [Fou02]**

Lähettävä agentti voi lähettää kaksi erilaista tiedustelupyyntöä: Query-if ja/tai Query-ref. Vastaanottava agentti joko hyväksyy (agree) tai hylkää (refuse) pyynnön. Mikäli vastaanottaja hyväksyy ”Query-if”-pyynnön, se vastaa joko tosi (true) tai epätosi (false). Mikäli vastaanottaja hyväkyy ”Query-ref”-pyynnön, se vastaa agentin suorituksesta saadun tuloksen. ”Failure”-viesti on tieto toiminnon epäonnistumisesta. [Fou02]

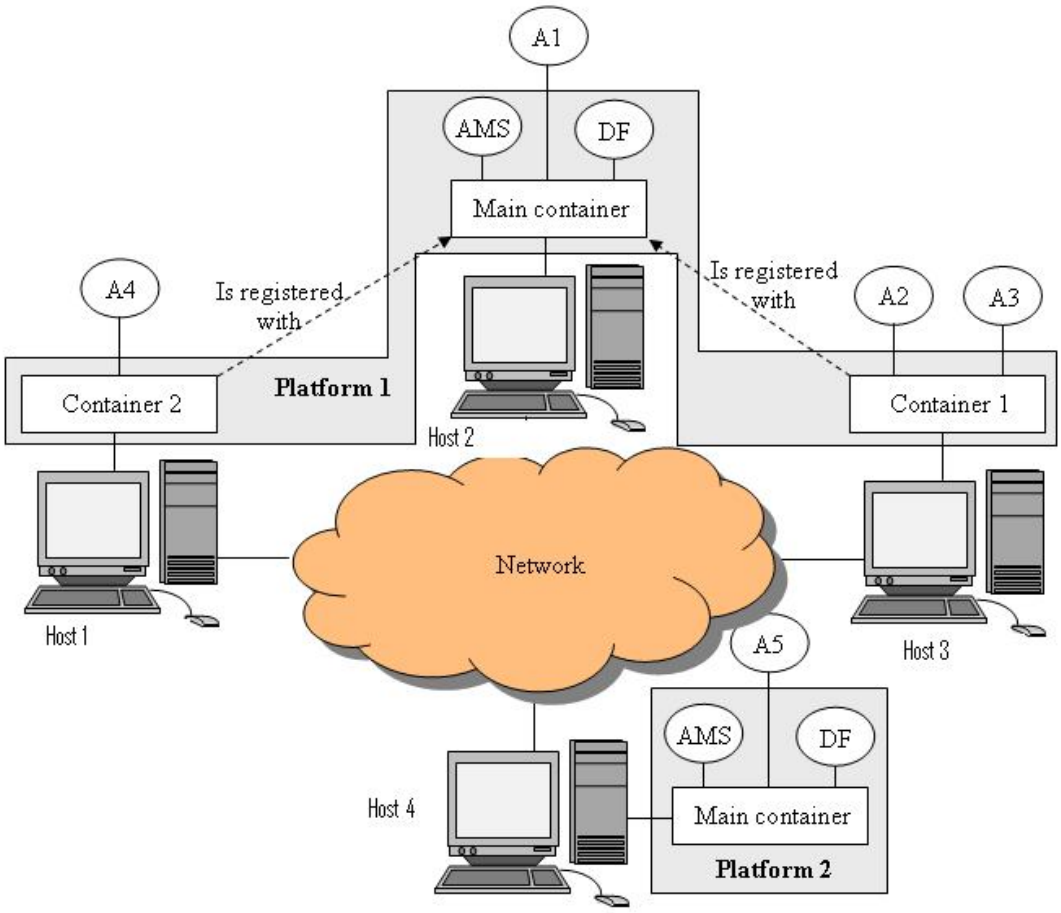
## 2.2 JADE-arkkitehtuuri

JADE on kommunikointiarkkitehtuuri, joka hallitsee agenttien välisiä ACL-viestijonoja (Agent Communication Language). ACL-viestit ovat yksityisiä jokaiselle agentille. JADE hyödyntää FIPA:n kehittämiä kommunikointimalleja. JADE-arkkitehtuuri voidaan jakaa korkeamman tason arkkitehtuuriin ja JADE:n sisäiseen arkkitehtuuriin. JADE:n korkeimman tason arkkitehtuuri jakautuu eri kerroksiin, joita ovat JADE-kerros ja Javan virtuaalikonekerros. JADE-kerros tarjoaa ohjelmistoagentit ja niiden säiliöt (container). Yhteen säiliöön voi kuulua monta agenttia. Säiliöt yhdistetään Java-virtuaalikerroksen avulla riippumatta laitteistosta, käyttöjärjestelmästä tai verkkotyypistä. Kuvassa 6 on rakennetta selventävä kuva. [BCP03]



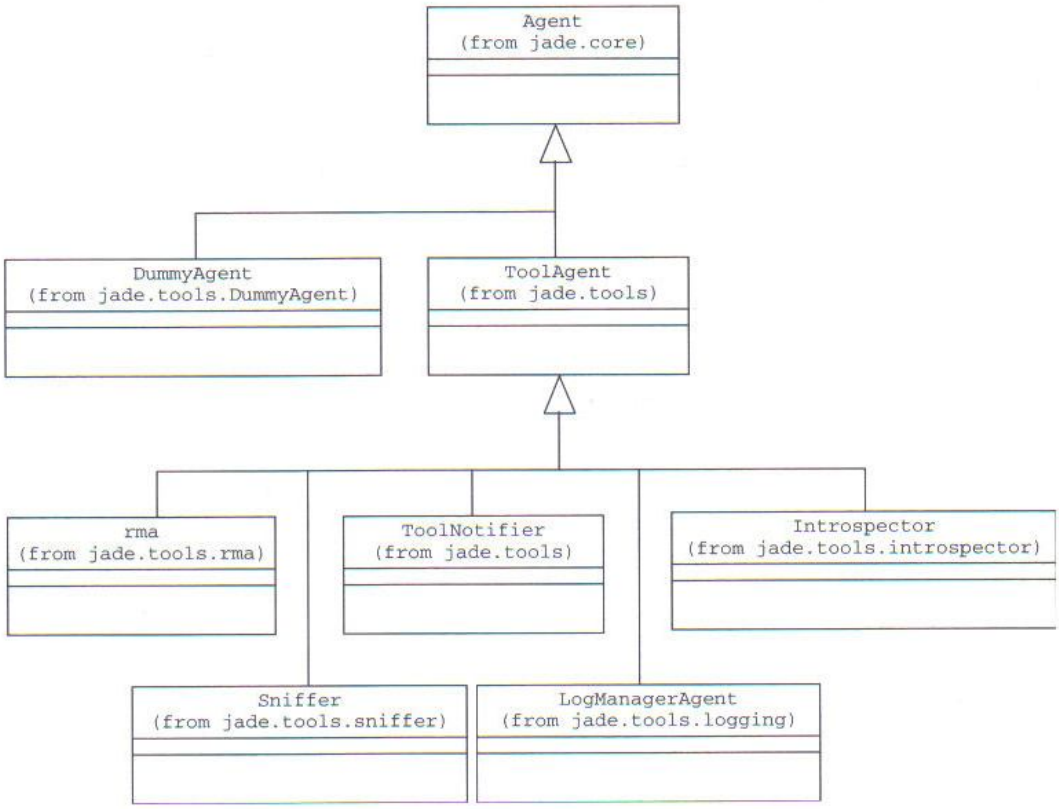

**Kuva 6. JADE-arkkitehtuurin kerrokset. [BCP03]**

Kuvassa 7 on esimerkki ohjelmistoagenteilla toteutetusta arkkitehtuurista. Se koostuu kahdesta erillisestä agenttialustasta, joilla on oma agenttienhallintajärjestelmä (AMS, Agent Management System) ja eri määrät säiliöitä, joissa ohjelmistoagentit sijaitsevat. [JAT10]




**Kuva 7. Esimerkki JADE-agenteilla toteutetusta arkkitehtuurista. [JAO10]**

JADE:n sisäinen arkkitehtuuri koostuu erilaisista luokista, joista jokaisella on oma tehtävänsä. Luokkien sijainti arkkitehtuurissa selviää hyvin sisäisen arkkitehtuurin luokkarakenteesta kuvasta 8.




**Kuva 8. JADE:n luokkarakenne. [BCG07]**

Jade.core on JADE-arkkitehtuurin ydin. Ohjelmistoydin sisältää kolme perusvalttia, jotka ovat: [BCG07]

1. Viestien välitys ja kapasiteetin jako.
2. Hajautettu ympäristö, joka tulee agenttialustaa.
3. Esiasennettu käyttäytyminen perustehtävien rakenteellisille tarpeille.

Dummyagent (jade.tools.Dummyagent) on yksinkertainen ja hyödyllinen työkalu viestien vaihtoon JADE-agenttien välillä. Se on monitorointiin ja virheiden etsimiseen tehty työkalu graafisella käyttöliittymällä. Graafisen käyttöliittymän avulla on mahdollista lähettää ACL-viestejä muille agenteille. Sen avulla voidaan myös tehdä aikajärjestyksessä listaus kaikista ACL-viesteistä, joita on lähetetty ja vastaanotettu.

ToolAgent (jade.tools) on abstrakti yläluokka agenttien työkaluille, joita ovat RMA, ToolNotifier, Introspector, LogManagerAgent ja Sniffer. RMA (jade.tools.rma) on JADE-agenttien hallintaan suunniteltu graafinen käyttöliittymä. ToolNotifier toimii siltana

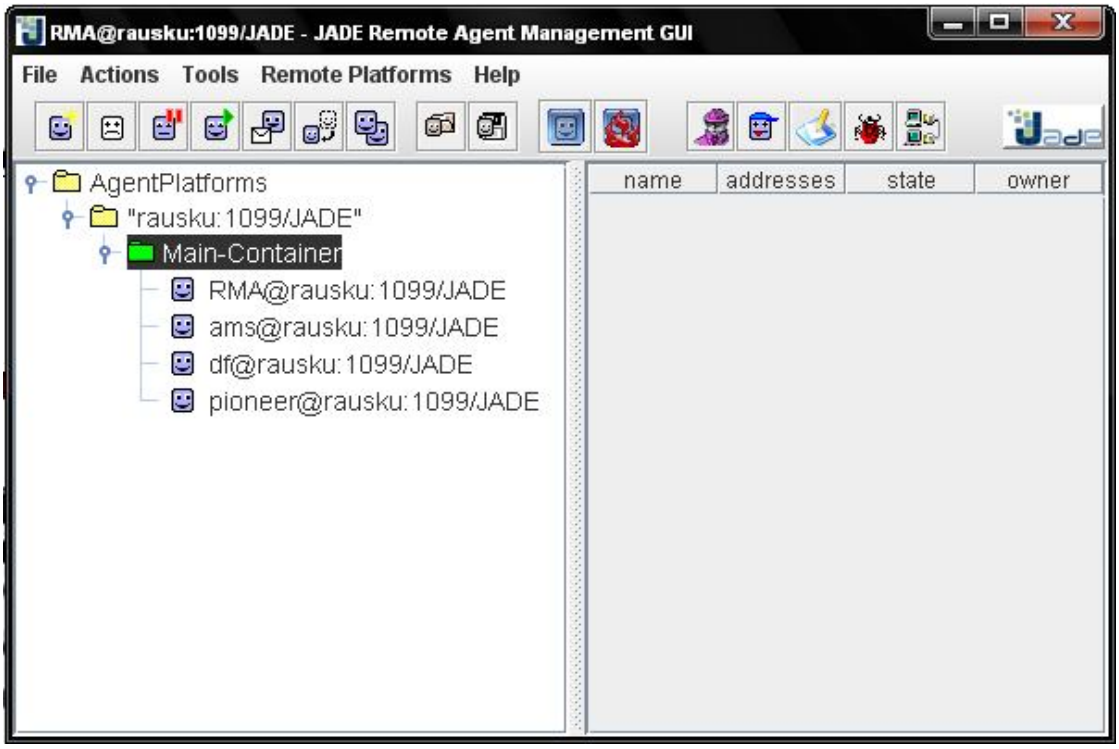
JADE:n tapahtumaluokkien ja ”jade.domain.introspection”-ontologian välillä. Tätä “ToolNotifier”-agenttia käyttää “Sniffer”- ja “Introspector”-agentit.

Introspector (tools.introspector) on agentti, joka monitoroi agentin elinkaarta. Se hoitaa ACL-viestien vaihdot ja käyttäytymisen agenttia suorittaessa. Tarvittaessa se laittaa joko lähetettäviä tai vastaanotettavia viestejä jonoon odottamaan suoritusta. LogManagerAgent (tools.logging) on agentti, jonka tehtävänä on tallentaa lokitiedot rekisteriin, josta niitä voi tarpeen vaatiessa tarkastella.

”Sniffer”-agentti (tools.sniffer) muodostaa agenttien välisten ACL-viestien kulusta UML-kaavioista (Unified Modelling Language) tutun sekvenssikaaviota muistuttavan kaavion. ”Sniffer”-agentti on oma Java-sovellus, joka jäljittää viestejä Java-ympäristössä. Se on hyvin integroitu Java-ympäristöön ja on myös erittäin hyödyllinen selvitettäessä eri agenttien välistä käyttäytymistä. ”Sniffer”-agentti mahdollistaa myös eri agenttien välisten ACL-viestien lukemisen.

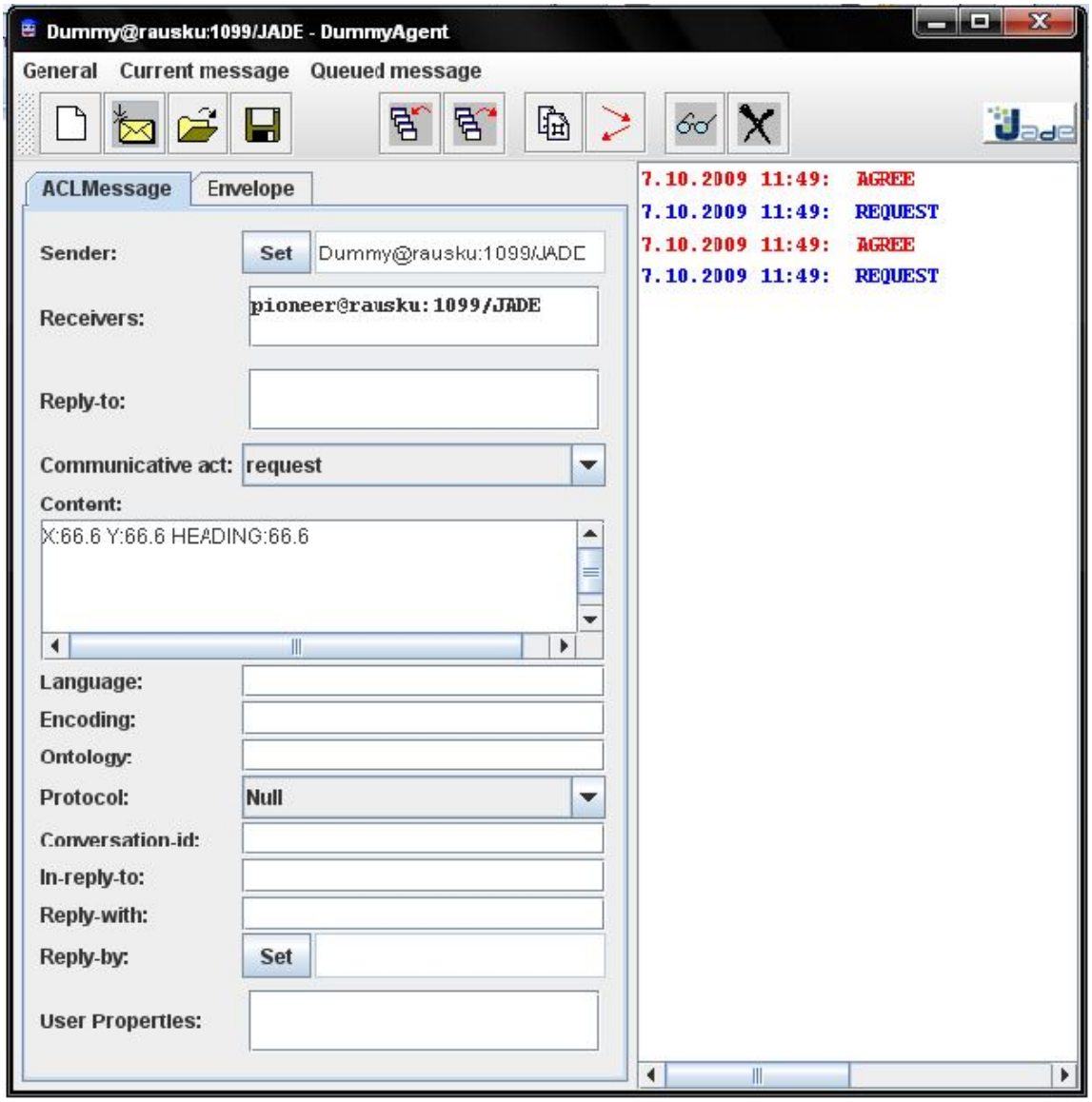
## 2.3 JADE:n käyttöliittymä

Agenttien lisäys ja poisto on helpointa JADE:n graafisessa ikkunassa. Kuitenkin kaikki toiminnot voidaan tehdä myös komentoriviltä. Ikkunassa on mahdollista luoda useita säiliöitä (container), joihin agentit luodaan. Säiliöt voivat koostua useasta eri agentista. Graafinen JADE-kehys helpottaa huomattavasti agenttien hallintaa, koska kaikki toiminnot voidaan tehdä kehyksen eri valikoista. Kuvassa 9 on graafisen JADE-kehyksen pääikkuna.



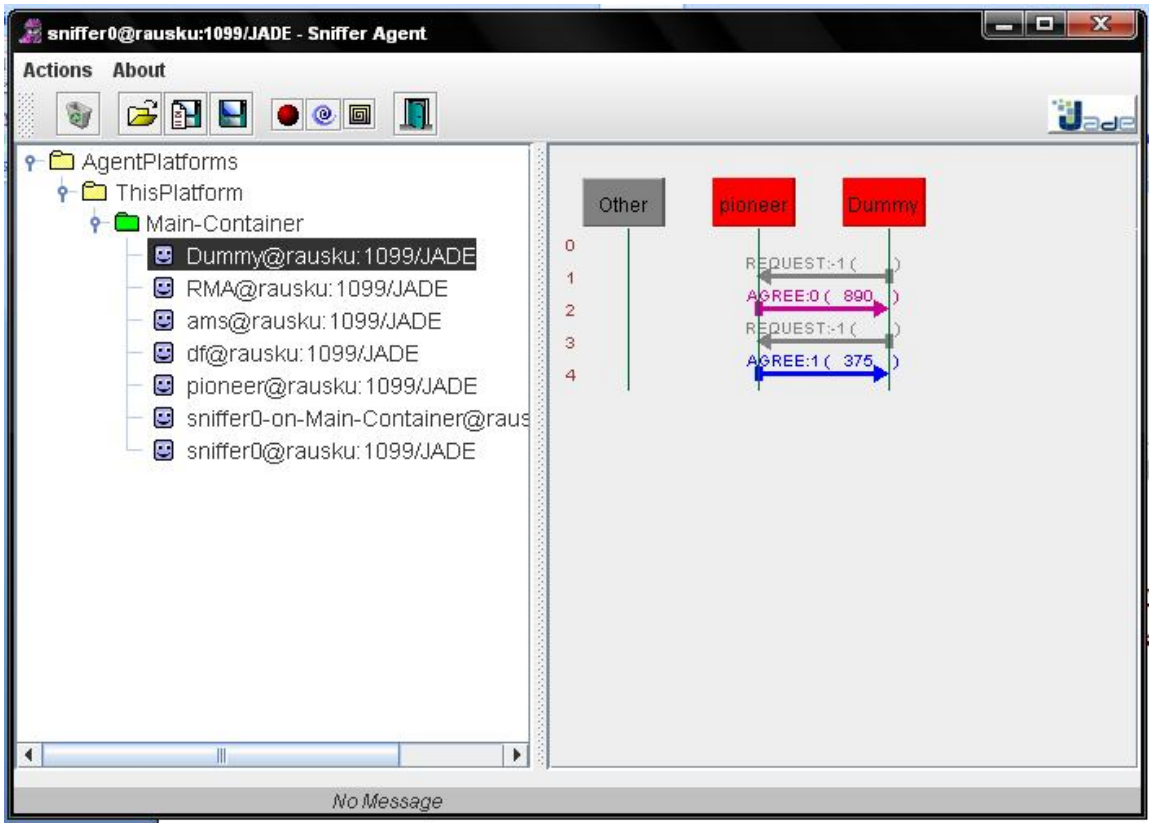

**Kuva 9. JADE-kehyksen graafinen käyttöliittymä.**

”Dummy”-agentti on viestien lähettämiseen agenttien välillä sopiva agentti. Sen käynnistäminen onnistuu pääikkunan valikoista. Käynnistetty ”Dummy”-agentti avaa ikkunan, jolla voi määritellä sen, mille agentille viestejä halutaan lähettää. Kuvassa 10 on ”Dummy”-agentin viesti-ikkuna, jolla viestejä lähetetään. Lähetysikkunassa määritellään viestin vastaanottaja(t). Tärkeää on myös määritellä se, minkä tyyppisestä viestistä on kysymys. Määrittely tapahtuu ”Communicative act”-valikosta. ”Content”-kenttään laitetaan lähetettävä viesti.




**Kuva 10. Dummy-agentin viesti-ikkuna.**

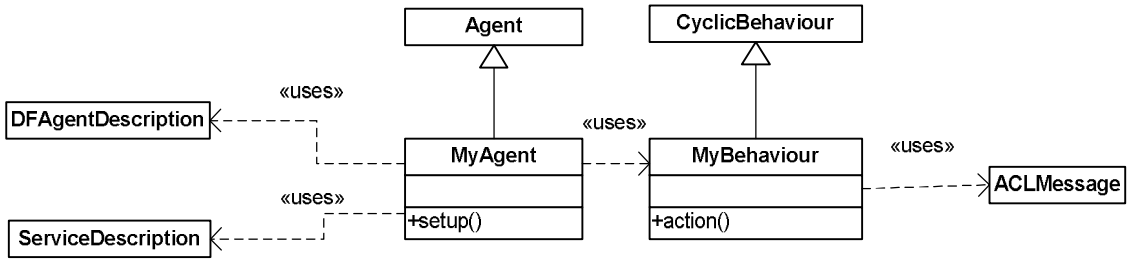
Eri agenttien välistä viestiliikennettä voidaan seurata käyttämällä ”Sniffer”-agenttia, joka näyttää lähetetyt viestit nuolina eri agenttien välillä. Eri agenttien välisiä viestejä on myös mahdollista katsella tarkemmin. ”Sniffer”-agentti käynnistetään vasta sitten, kun viestin lähettävä ”Dummy”-agentti ja viestin vastaanottava agentti on käynnistetty. Kun ”Sniffer”-agentti on käynnistetty, valitaan seurattavat agentit. Tämän jälkeen lähetetään ”Dummy”-agentilla viesti, jotta nähdään, kulkeeko se halutulla tavalla. Viestiliikenne näkyy tällöin ”Sniffer”-agentin graafisessa ikkunassa nuolina eri agenttien välillä. Viestin tekstistä näkyy myös se, onko viesti hyväksytty (agree) vai ei-hyväksytty (not agree). Kuvassa 11 on esimerkki ”Sniffer”-ikkunasta.




**Kuva 11. Sniffer-ikkuna.**

## 2.4 JADE-agenttien sisäinen rakenne

JADE-agentti muodostuu ”MyAgent”-luokasta, jonka voi nimetä käyttötarkoitukseen sopivammalla nimellä. ”MyAgent”-luokan sisällä on agentin käyttäytymistä määrittelevä luokka MyBehaviour. MyAgent ja MyBehaviour ylikirjoittavat yliluokkansa kuvan 12 kaavion mukaisesti. MyAgent sisältää ”setup”-metodin, johon määritellään kyseisen agentin tunnistetiedot. ”Setup”-metodissa luodaan agentista ”DFAgentDescription”- ja ”ServiceDescription”-oliot.




**Kuva 12. JADE-agentin sisäinen rakenne.**

”ServiceDescription”-olio sisältää agentin tunnistetietoja ja sen eri metodeita ovat setType, setName, setOwnership ja addOntologies. SetType asettaa palvelun tyypin, setName palvelun nimen, setOwnerShip omistussuhteen ja addOntologies ontologian nimen. ”ServiceDescription”-luokka sisältää myös muita metodeja, jotka parhaiten löytyvät JADE-paketin mukana tulevasta API-kirjastosta (Application Programming Interface). Näistä ”DFAgentDescription”-luokka implementoi FIPA-agenttien hallinnan mukaisen kuvauksen agentista. MyBehaviour sen sijaan sisältää ”action”-metodin, jossa on agentin toiminnallisuuteen liittyvä ohjelmakoodi. Yksinkertaisimmillaan on vain ”ACLMessage”-luokka, jolla hallitaan agentin viestitoiminnot.

## 2.5 Yhteenveto

JADE-ohjelmistoagentit perustuvat FIPA-standardointijärjestön protokolliin. Agenttien tehtävänä on olla Java-kielisiä ohjelmiston palasia, jotka voivat olla joko täysin itsenäisiä tai ohjelmiston käyttäjän hallinnassa. JADE-agenttien kehitysympäristö sisältää agenttien ajamiseen, viestimiseen ja tilojen seuraamiseen sopivat työkalut. Agenttiympäristöt koostuvat säiliöistä, joissa voi olla useita agentteja.

FIPA:n määrittelemä ohjelmistoagenttiarkkitehtuuri perustuu protokolliin. Niiden avulla mahdollistetaan ohjelmistoagenttien välinen kommunikointi. Agenttien ohjausjärjestelmä huolehtii siitä, että viestit kulkevat oikeiden ohjelmistoagenttien välillä. JADE-ohjelmistoagentteja on helpoin hallita JADE:n oman graafisen ohjelmiston avulla. Sillä on mahdollista luoda, poistaa ja muokata JADE-ohjelmistoagentteja. Viestien lähettäminen ja viestiliikenteen seuraaminen on myös mahdollista.

# 3 PIONEER-ROBOTTI

Tässä luvussa käsitellään aluksi robotteja yleisesti ja sen jälkeen tutustutaan tarkemmin Pioneer P3-DX -robottiin. Luvussa 3.1 perehdytään robotiikan historiaan, nykyhetkeen ja tulevaisuuteen. Luku 3.2 muodostaa käsityksen robotin liikkumisesta ja kulkureittien määrittelystä. Luvussa 3.3 kerrotaan Pioneer-robotin ominaisuuksista, toiminnasta ja siihen liittyvistä projekteista. Lopuksi luvussa 3.4 on yhteenveto luvun 3 sisällöstä.

## 3.1 Robotiikka

Robotilla tarkoitetaan sähkömekaanista laitetta, joka toimii fyysisessä maailmassa. Robottien tehtävänä on helpottaa ihmisen toimintaa joko suoraan ihmisen komennossa, osittain ihmisen komennossa tai täysin autonomisesti tietokoneen komennossa. Robotiikkaan liittyvien tekniikoiden kehitys on ollut nopeaa ja robotit ovat tulleet ihmisille paljon aiempaa arkipäiväisemmiksi mm. palvelurobotiikan ansiosta. Suurin osa roboteista tehdään silti edelleen teolliseen käyttöön. Niillä on nykyään erittäin tärkeä tehtävä monien yrityksien kilpailukyvyn parantamisessa ja lisäksi myös tehtäviä, joita ei pystyttäisi suorittamaan ilman robotteja. Täytyy kuitenkin muistaa, että robottien käytössä tärkein toimija on ihminen, joka suunnittelee, kokoaa ja ohjelmoi laitteet sekä huolehtii niiden kunnossapidosta. [Rob99, Wik13]

Robottien kehitys alkoi jo vuonna 1738, jolloin ranskalainen Jacques de Vaucanson rakensi ensimmäisen robotin, joka osasi soittaa huilua. Hän rakensi myös mekaanisen ankan, joka söi jyviä ja ulosti. Myöhemmin vuonna 1898 Nikola Tesla kehitteli radio-ohjattavan veneen, jota hyödynsi myöhemmin Yhdysvaltojen laivasto rakentamalla tekniikan pohjalta torpedon. Ensimmäinen robottilaite, autopilotti, valmistui vuonna 1913 ja ensimmäinen robotin lausuma sana kuultiin vuonna 1923 näytelmässä RUR. Ensimmäinen teollisuusrobotti otettiin käyttöön USA:ssa vuonna 1962. Teknologian kehityksen myötä teollisuusrobotit yleistyivät 70-luvulla, lähinnä autoteollisuudessa, jossa tärkein käyttökohde oli hitsaus. Robottien kehityksestä ja niiden käytön räjähdysmäisestä lisääntymisestä kertoo se, että vuonna 1980 maailmassa oli noin 8000 teollisuusrobottia ja jo vuonna 1995 niitä oli noin 650 000. Suomessa robotisointi käynnistyi 70-luvulla,

jolloin pääpaino oli maalausrobotiikassa. 80-luvulla robotiikan sovelluksia alettiin käyttää Suomessakin laajemmin, muun muassa hitsauksessa ja kappaleenkäsittelyssä. Suomessa vuosina 1984–1987 kaikista robotiikan sovelluksista oli hitsaukseen liittyviä noin puolet ja robotteja kokonaismäärältään yhteensä noin 500 kappaletta. Vuosina 1988–1991 kehiteltiin runsaasti uusia sovelluksia esimerkiksi lääketieteen piirissä, muun muassa laserleikkaus. Robotteja oli tuolloin Suomessa yhteensä noin 1000 kappaletta, kun vuonna 1996 vastaava luku oli noin 1650. Vuonna 2000 robottien määrä oli noussut Suomessa 3193:een ja vuonna 2007 niitä oli jo 5821 kappaletta. [Rob99, Wik13]

Kansainvälisen Robottiyhdistyksen mukaan robotti on ohjelmoitavissa oleva vähintään kolminivelinen mekaaninen laite, jonka tehtävänä on liikuttaa kappaleita, osia tai työkaluja teollisuuden erilaisissa ohjelmoitavissa sovelluksissa. Nykyään pelkkä uudelleenohjelmoitavuus ei riitä, vaan nykyaikaisissa aistiohjatuissa robottisovelluksissa robotit on saatava muodostamaan omat liikeratansa tuotteiden suunnittelutiedoista ja ympäristömallista. Liikeratoja päivitetään prosessia tarkkailevien anturien avulla. [Lah08]

Ensimmäiset sähkömekaaniset robotit olivat erittäin sovelluskohtaisia. Niitä voitiin käyttää vain siihen yhteen tarkoitukseen, jota varten ne oli rakennettu. Koneiden asetusten säätäminen oli todella työlästä ja usein, jos työstettävää kappaletta piti muuttaa, täytyi uusia koko robotti. Robottien siirtyminen pienempien sarjojen valmistukseen loi pohjan ohjelmallisesti muunneltaville roboteille. Ohjelmallisesti muunneltavat robotit lisäsivät joustavuutta tuotantoon ja mahdollistivat aiempaa paremman vastaamisen asiakkaan tarpeisiin. [Rob99, Wik13]

Ohjelmallisesti muunneltavat robotit yleistyvät myös, koska niiden ohjelmointikielet kehittyivät merkittävästi. Nämä niin sanonut korkean tason kielet nopeuttivat ja helpottivat robottien ohjelmien tekoa merkittävästi. Tärkeää oli myös anturitekniikoiden kehitys, mikä mahdollisti automaattisesti ympäristöön sopeutuvien älykkäiden robottijärjestelmien kehittämisen entistä vaativampiin sovelluksiin. Nykyisin robotin liikerata voidaan määrittää kokonaan etukäteen, valita toimintaympäristön tapahtumien perusteella tai luoda antureiden ja robotin aiempien liikkeiden avulla. [Rob99, Wik13]

Teollisuusrobotin tehtävät voidaan jakaa tehtävänkuvan mukaan. Pääasiassa niitä käytetään kolmella eri osa-alueella, joita ovat maalaus-, prosessi- ja kokoonpanorobotit. Maalausrobotit ovat nopeita ja niiden toiminnassa on tärkeää lähinnä maalaustuloksen tasaisuus, ei niinkään tarkkuus. Prosessirobotit sen sijaan tekevät tarkasti jotain tiettyä yksit-

täistä toimintoa, mutta eivät ole helposti sovellettavissa muihin tehtäviin. Kokoonpanorobotit taas ovat työjäljessä sekä erittäin tarkkoja että nopeita, mutta niillä on alhainen kappaleenkäsittelykyky. [Rob99, Wik13]

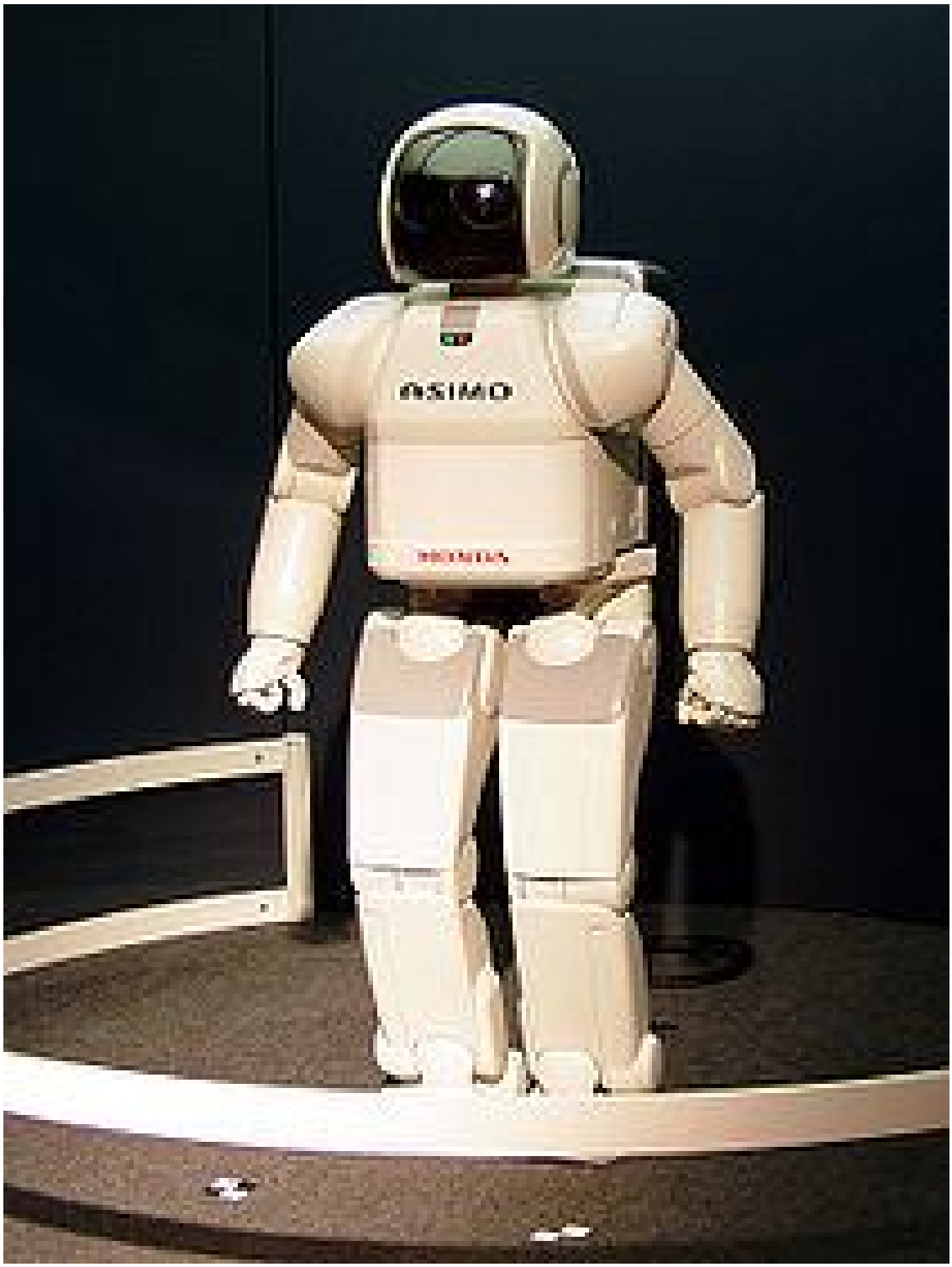
Teollisuusrobotit ovat usein nivelrobotteja, jotka matkivat ihmisen käsivarsien liikkeitä ja mahdollistavat samankaltaisten tehtävien teon kuin ihmiskädellä. Niiden nopeus on kuitenkin moninkertainen ja tarkkuus lähes virheetön ihmisen tekemään työhön verrattuna. Robotin rakenne ei juuri vaikuta sen tarkkuuteen, vaan suurempi merkitys on käyttökohteella, josta riippuen robotin tarkkuus on +/- yksi millimetri, tosin usein tarkkuus on nykyisin vielä paljon parempi. Kokoonpanorobotti on kaikkein tarkin, sillä sen on pystyttävä +/- 0.005-0.1 millimetrin asemointitarkkuuteen. Robotit pystyvät käsittelemään massaltaan hyvin vaihtelevia kappaleita. Pienimmät robotit käsittelevät 0-5 kilogrammaa painavia kappaleita, mutta suurimmat kykenevät siirtämään jopa tuhansien kilogrammojen painoisia kappaleita. [Rob99, Wik13]

Teollisuusrobottien lisäksi on olemassa myös paljon muita erilaisia erikoisrobotteja. Yleisimpiä ovat niin sanotut mobiilirobotit, jotka ovat tietokoneen ohjaamia ajoneuvoja ja työkoneita. Niitä on ollut käytössä jo 1970-luvulta lähtien erityisesti varastojen ja teollisuuden eri käyttökohteissa. Kyseisiä robotteja hyödynnetään muun muassa rakentamisessa, kuljetuksessa, kaivoksissa, palontorjunta- ja pelastustehtävissä, vartioinnissa, maa- sekä metsätaloudessa, siivouksessa, satelliittien huolto- ja kokoonpanotehtävissä sekä sodankäynnissä. GPS-satelliittipaikannus (Global Positioning System) on tullut erittäin tärkeäksi osaksi robottien navigointijärjestelmiä.

Palvelurobotiikka on yksi nopeasti kehittyvä valtavan potentiaalin omaava kasvuala. Palvelurobotiikalla tarkoitetaan robotteja, jotka hoitavat palvelutehtäviä. Tällaisia tehtäviä voivat olla esimerkiksi lentokoneiden peseminen, lehmien koneellinen lypsäminen, pölyjen imeminen ja ruohon leikkaaminen.

Kaikkein kehittyneintä ja robottien valmistajille haasteellisinta robottityyppiä ovat androidit. Niiden tarkoituksena on matkia ihmisen ruumiinrakennetta, jotta ne voivat toimia ja liikkua ihmisten maailmassa. Esimerkiksi Japanissa on kehitetty ihmistä lyhyempi androidi, jolla on näköaisti ja joka pystyy liikkumaan ihmisten tavoin. Androideja on ajateltu voitavan hyödyntää ihmiselle liian vaarallisissa töissä ja tulevaisuudessa onkin mahdollista, että ne voivat jollain alalla syrjäyttää ihmisen kokonaan. [Wik13]

Yksi kuuluisimmista androideista on Hondan kehittämä 1.3 metriä korkea ASIMO (kuva 13). Se on kooltaan ihmistä pienempi, mutta se osaa monia ihmiselle ominaisia toimintoja, kuten esimerkiksi käveleminen, juokseminen ja portaiden nouseminen. Lisäksi ASIMO osaa sanoa ihmiselle käsipäivää, potkaista jalkapalloa ja työntää edessään vaunuja. [Wiki13]




**Kuva 13. ASIMO. [Wiki13]**

Tulevaisuuden robotit voivat olla erittäin pieniä ja niitä kutsutaankin nanoroboteiksi. Erityisesti lääketeollisuus on kiinnostunut kehittämään nanorobotteja, jotka voisivat esimerkiksi puhdistaa haitalliset bakteerit, hammaskiven ja ruuanmurut suusta. Nanorobotteja voisi myös laittaa ihmisen verenkiertoon, jossa robotit etsisivät verisuonten seinämiltä kolesteroliplakkeja ja söisivät ne pois puhdistaen suonien sisäpinnan. Myös syöpäsoluja tuhoavia nanorobotteja on kehitteillä. Nanorobotit ovat tosin vielä teoria- ja kokeiluasteella: todennäköisesti tulee kulumaan vielä useita vuosia, ennen kuin niistä on tarjolla käytännön sovelluksia. [Enq99]

Robotin ohjaamiseen on mahdollista käyttää myös limasientä, jota voidaan käyttää robotin "aivoina". Physarum polycephalum -limasieni kykenee paikallistamaan ruoan ja liikkumaan sen luo. Se myös välttää valoa ja viihtyy kosteissa paikoissa. Limasieni saatiin ohjaamaan robottia, kun kuusijalkaisen robotin päälle kasvatettiin kuusisakarainen limasienikasvusto ja kytkettiin se tietokoneella robottiin. Kun robotin jalka on valossa, välittyy siitä tieto tietokoneen kautta limasienisakaran päällä olevaan valonlähteeseen, joka syttyy. Valo taas saa limasienen antamaan robotille käskyn siirtää jalkaa ja saa täten sienen ohjaamaan robotin pois valosta. Käyttökohteina tällaisella robotilla voivat olla pimeät, kosteat ja epämääräisen muotoiset paikat, joissa robotin ohjaaminen pelkällä tietokoneella on hankalaa. Jatkossa tarkoituksena olisi kyetä kasvattamaan limasieni suoraan robotin päälle. Limasienen ohjaaman robotin kehittäjien mukaan tällainen "biologinen ohjelmointi" voisi olla tietokonetta yksinkertaisempi tapa hallita robotteja. [New06]

## 3.2 Liikkuminen ja kulkureitin määrittely

Liikkuvan robotin yksi tärkeimmistä tehtävistä on havainnoida ympäristöään antureillaan. Niistä saamallaan tiedolla robotti osaa paikallistaa sijaintinsa. Täysin virheetöntä karttaa on robotin ainakin vielä melko mahdotonta tuottaa. Mitä enemmän tarkkuutta vaativasta tehtävästä on kyse, sitä tarkempi pitää robotin muodostama kartta olla ja tämä asettaa omat haasteensa robottien kartoitusalgoritmeille.

Robotin sijainnin määrittelyyn voidaan käyttää kameraa, laseria, infrapunaa, tutkaa tai kompassia. Myös niiden yhdistelmiä on mahdollista käyttää. Näitä erilaisia ympäristöä

analysoivia laitteita hyödyntäen voidaan muodostaa kartta, josta saadaan mahdollisimman luotettava. Aina ennalta-arvattavan kartan muodostus ei ole mahdollista, vaan eteen saattaa tulla yllättäviä esteitä. Näihin on syytä varautua niin, että robotti osaa havainnoida odottamattomat esteet ja tehdä toimenpiteet törmäyksen estämiseksi. [HSB03]

Robotin kartan määrittelyyn vaikuttaa myös ympäristö, jossa robotin on tarkoitus liikkua. Helposti määritettäviä muotoja ovat suorat seinät, seinien leikkauspisteet ja paikannettavat kiinteät esteet. Ongelmia syntyy, mikäli paikannettava tila on epämääräisen muotoinen: esim. seinät ovat rosoiset ja suoria pintoja ei ole. Tällaisia tiloja ovat esimerkiksi rauniot, maasto ja luolat. Myös lasiseinät voivat olla robotin käytön kannalta hankalia varsinkin, jos paikkatietojen määritykseen käytetään kaikuluotausta. Robotin suunnan ja tarkan sijainnin tietäminen helpottaa kartan muodostamista ympäristöstä. [HoK96, Tha97, Thr02]

Robottiin asennettavia sensoreita on monenlaisia, esimerkiksi kaikuluotain, laserskanneri, infrapunamittari, tutka, kamera tai GPS-paikannin. Näiden kaikkien heikkoutena on se, että ne ovat alttiita mittausvirheille eli kohinalle. Lisäksi ääneen ja valoon perustuvien sensoreiden ongelmana on se, etteivät niiden tuottamat ääni ja valo läpäise seiniä. Robotin liikkumisesta voi myös tulla mittavirheitä, mikä vaikeuttaa robotin virheetöntä paikantamista. [Thr02]

## 3.3 Pioneer-robotti

Pioneer P3-DX on maailman suosituin tutkimusrobotti, jonka monipuolisuus, luotettavuus ja kestävyys tekevät siitä vertaansa vailla olevan. P3-DX on täysin ohjelmoitavissa ja kestää kovan luokkahuone- sekä laboratoriokäytön. Robotti on valmistettu kestävästä alumiiniseoksesta. Siinä on 19 senttimetriä halkaisijaltaan olevat renkaat ja kahdeksan ultraäänianturia, joita voi käyttää kohteiden havainnointiin. Robotin (kuva 14) huippunopeus on 1.6 m/s ja se voi kantaa maksimissaan 23 kilogrammaa painavaa esinettä. [Ade12]


**Kuva 14. Pioneer P3-DX -robotti. [Ade12]**

Robotin liikkuminen perustuu siis ultraääneen, jonka avulla se mittaa etäisyyksiä kohteisiin. Siinä on itsessään kiinni anturit ultraäänen lähettämiseen. Robotille on myös mahdollista lähettää ohjaussignaalia hyödyntäen verkkoyhteyksiä tai sarjaporttia. [Mob07]

Robotti on hankittu Itä-Suomen yliopistolle konenäön ja tekoälyn kehittämistä varten. Niiden avulla robotista olisi mahdollista saada entistä itsenäisemmin liikkuva, sillä se pystyisi itse havaitsemaan esteitä ja väistämään niitä. Tällä hetkellä projekti on siinä vaiheessa, että robottia ohjataan ohjelmistoagentin avulla. Ohjelmistoagentilla lähetetään robotille viesti, jonka mukaisesti robotti liikkuu. Tulevaisuudessa viestin voisi lähettää jokin konenäköä hyödyntävä anturi.

Robotti sisältää oman C++-kielellä toteutetun ARIA-rajapinnan, jota voidaan hyödyntää tekemällä omia sovelluksia. Pioneer-robotilla on olemassa myös oma MobileRobotsin kehittelemä sovellus robotin karttatietojen määrittelyyn. Tämän tutkielman testiajoissa

on kuitenkin hyödynnetty FT Mauno Rönkön C++:lla koodaamaa erillistä robotin ohjausrajapintaa, jolle lähetetään ohjausviestejä JADE-ohjelmistoagentin avulla. Ohjausviestit lähetetään hyödyntäen tietokoneen sarjaporttia. [Mob07]

Robotille lähetetään koordinaatit ja suuntakulma. Se liikkuu näitä tietoja hyödyntäen haluttuun kohteeseen. Vielä tässä vaiheessa robottia ohjaavalle ”Pioneer”-agentille on lähetettävä koordinaatit hyödyntäen ”Dummy”-agenttia, joka on yksinkertainen viestinlähetysohjelmistoagentti. Viestien kulkua eri ohjelmistoagenttien välillä pystytään seuraamaan ”Sniffer”-ohjelmistoagentilla, joka näyttää agenttien väliset kommunikaatiot viivoilla. Viivoista on mahdollista katsoa kyseisen viestin sisältö. Tarkemmin viestien lähettämisestä kerrottiin luvussa 2.3.

## 3.4 Yhteenveto

Robottien kehitys on ollut erittäin nopeaa ja se onkin hyvä osoitus ihmisen innovatiivisuudesta sekä halusta automatisoida töitään. Niiden kehityksen katsotaan alkavan jo 1738, jolloin ranskalainen Jacques de Vaucanson rakensi robotin, joka osasi soittaa huilua. Robottien kehitys oli aluksi hidasta. Vasta 1970-1980-luvulla robottien kehitys nopeutui merkittävästi.

Robotiksi mainitaan usein sähkömekaaninen laite, joka toimii fyysisessä maailmassa. Kansainvälinen Robottiyhdistys on tarkentanut määritelmää siten, että robotti on ohjelmoitavissa oleva vähintään kolminivelinen mekaaninen laite. Sen tehtävänä on liikuttaa kappaleita, osia tai työkaluja teollisuuden erilaisissa ohjelmoitavissa sovelluksissa. Robotti päivittää liikeratojaan omien ympäristöä havainnoivien anturien avulla. Ne ovat muuttuneet vuosien saatossa yhteen sovelluskohteeseen räätälöidystä robotista moneen eri kohteeseen ohjelmoitavaksi robotiksi. Kehitystä on nopeuttanut ohjelmointikielien nopea kehitys. Teollisuusrobotit jaetaan tehtäväkuvan mukaan kolmelle eri osa-alueelle joita ovat maalaus-, prosessi- ja kokoonpanorobotit.

Teollisuusrobottien lisäksi on olemassa erilaisia erikoisrobotteja. Niitä ovat mm. mobiili-, palvelu- ja nanorobotit. Mobiilirobotit ovat tietokoneohjattuja ajoneuvoja ja työkoneita. Niitä hyödynnetään muun muassa rakentamisessa, kuljetuksessa, kaivoksissa, palontorjunta- ja pelastustehtävissä, vartioinnissa, maa- sekä metsätaloudessa, siivouk-

sessa, satelliittien huolto- ja kokoonpanotehtävissä sekä sodankäynnissä. Erilaiset palvelurobotit ovat myös kovasti yleistymässä. Niiden tehtävänä on esimerkiksi lentokoneiden peseminen, lehmien koneellinen lypsäminen, pölyjen imeminen ja ruohon leikkaaminen.

Ihmisen liikkeitä matkivia robotteja on myös yritetty kehittää. Tämä kaikista haasteellisin robottityyppi muistuttaa ulkonäöltään ihmistä. Näitä robotteja sanotaan androideiksi ja niistä on olemassa näkevä ja ihmisen tapaan liikkuva malli. Androideja olisi tarkoitus hyödyntää mahdollisesti ihmiselle vaarallisissa töissä.

Robotit voivat olla myös erittäin pieniä, niin sanottuja nanorobotteja. Lääketeollisuus on erityisen kiinnostunut niiden kehittämisestä. Niiden avulla voitaisiin kulkea ihmisen verisuonien sisällä ja parantaa erilaisia sairauksia.

Erilaiset anturit mahdollistavat robotin liikkumisen. Robotti havainnoi anturien avulla ympäristöään ja paikallistaa sijaintiaan. Se voi käyttää sijaintinsa määrittelyyn kameraa, laseria, tutkaa tai kompassia. Myös GPS-paikannus on mahdollinen. Mittausvirheet vaikeuttavat robotin liikkumista.

Ohjelmistoagenttien testaamiseen käytettiin Pioneer P3-DX -robottia, joka on maailman suosituin tutkimusrobotti. Robotin liikkumiseen käytetään ultraääntä, joka mahdollistaa esteiden väistelyn. Tässä tutkielmassa ohjelmistoagentti lähettää koordinaatit ja suuntakulman, joiden avulla robotti osaa liikkua. Vielä tässä vaiheessa koordinaatit tarvitsee lähettää tietokoneella ohjelmistoagentin avulla, mutta tulevaisuudessa olisi mahdollista kehittää järjestelmä, joka hyödyntäisi myös konenäköä koordinaattien määrittelyssä.

# 4 TESTIJÄRJESTELYT JA TULOKSET

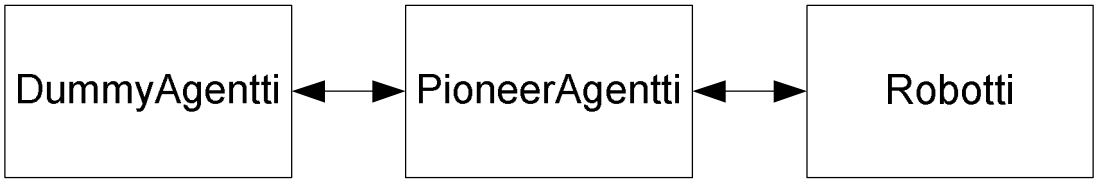
Pioneer P3-DX -robotilla suoritettiin testiajot, joiden järjestelyihin, toteuttamiseen ja tuloksiin perehdytään tässä luvussa. Luvussa 4.1 kerrotaan ongelman kuvauksesta. Luvussa 4.2 keskitytään Pioneer-agentin ohjelmointiin ja ohjelman rakenteeseen sekä kerrotaan ohjelmistoagenttien välisestä viestinnästä ja niiden eri protokollista. Luku 4.3 sisältää kuvauksen Pioneer-robotilla suoritettujen testiajojen alkuvalmisteluista, toteutuksesta ja tuloksista. Lopuksi luku 4.4 sisältää yhteenvedon koko luvun 4 sisällöstä.

## 4.1 Ongelman kuvaus

Tavoitteena on käyttää ohjelmistoagentteja Pioneer P3-DX -robotin ohjaukseen. Robotista on tarkoitus tehdä itsenäisesti toimiva kokonaisuus, jonka vuoksi sen tulisi osata havainnoida omaa ympäristöään. Tämä tullaan toteuttamaan soveltamalla konenäköä robotin ohjauksessa.

Tässä tutkielmassa perehdytään ainoastaan alimman tason agenttiin, joka huolehtii robotin liikkumisesta annettuun koordinaattiin. Robotin ohjauksesta vastaavalle agentille lähetetään koordinaatit ja robotti liikkuu niiden mukaisesti uuteen kohteeseen. Robotilta on myös mahdollista kysyä sen nykyistä sijaintia.

Koordinaattien lähettämiseen agentille käytetään JADE-kehystä, jolla luodaan viestien lähetykseen tarkoitettu ”Dummy”-agentti. JADE-kehys sisältää graafisen sovelluksen, jolla on helppo seurata viestien kulkua. Myös agentin sijainnin kysyminen toteutetaan ”Dummy”-agentin avulla. Jatkossa muut agentit voivat kommunikoida ohjausagentin kanssa suoraan eli silloin ”Dummy”-agenttia ei enää tarvita. ”Dummy”-agentin ja robotin välissä on ”Pioneer”-agentti, joka hoitaa viestien välityksen ”Dummy”-agentin ja robotin oman ohjainjärjestelmän välillä. Rakenne on kuvattu kuvassa 15.

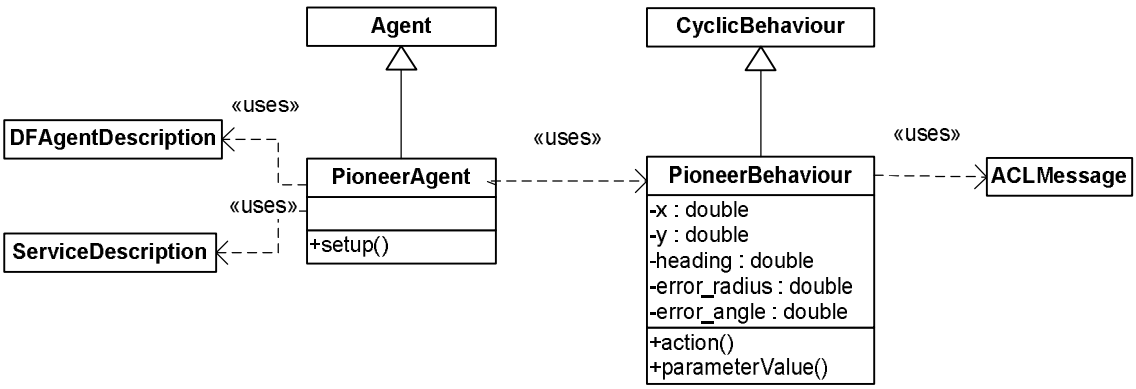



**Kuva 15. Kaavio robotin ja ohjelmistoagenttien välisestä viestinnästä.**

## 4.2 Pioneer-agentin ohjelmointi

”Pioneer”-agentti ylikirjoittaa ”Agent”-luokan ja ”Pioneer-agent”-luokkaan tulee pakollisena ”setup”-metodi, johon määritellään tarkemmin agentin tunnistetiedot. Luokka sisältää lisäksi ”DFAgentDescription”- ja ”ServiceDescription”-luokan. ”DFAgentDescription”-luokka implementoi FIPA-agentien hallinnan mukaisen kuvauksen agentista. ”ServiceDescription”-luokan eri metodeita ovat setType, setName, setOwnership ja addOntologies. SetType asettaa palvelun tyypin, setName palvelun nimen, setOwnerShip omistussuhteen ja addOntologies ontologian nimen. ”ServiceDescription”-luokka sisältää myös muitakin metodeja, jotka parhaiten löytyvät JADE-paketin mukana tulevasta API-kirjastosta.

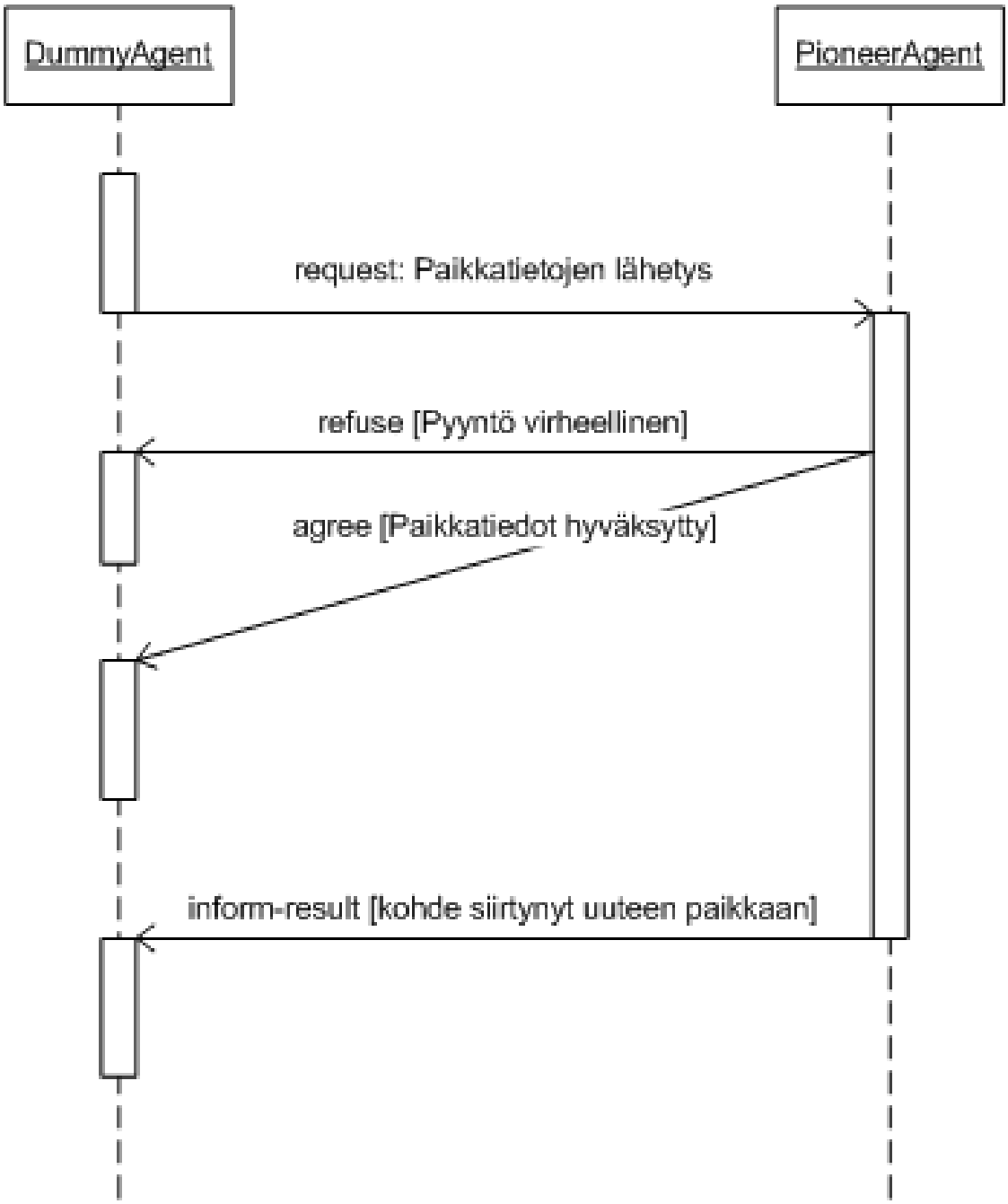
Pioneer-agentille luodaan käyttäytyminen erillisellä ”PioneerBehaviour”-luokalla. Se toimii ”Pioneer-agent”-luokan sisäluokkana. PioneerBehaviour ylikirjoittaa ”CyclicBehaviour”-luokan. PioneerBehaviour luokassa on ”action”-metodi, johon sisällytetään kaikki agentin toiminnallisuuteen liittyvä ohjelmakoodi. Se käyttää viestien lähetyksessä ”ACLMessage”-luokkaa. Tarkemmin luokkien välisiä yhteyksiä selventävä luokkakaavio esitetään kuvassa 16.




**Kuva 16. ”Pioneer”-agentin luokkakaavio.**

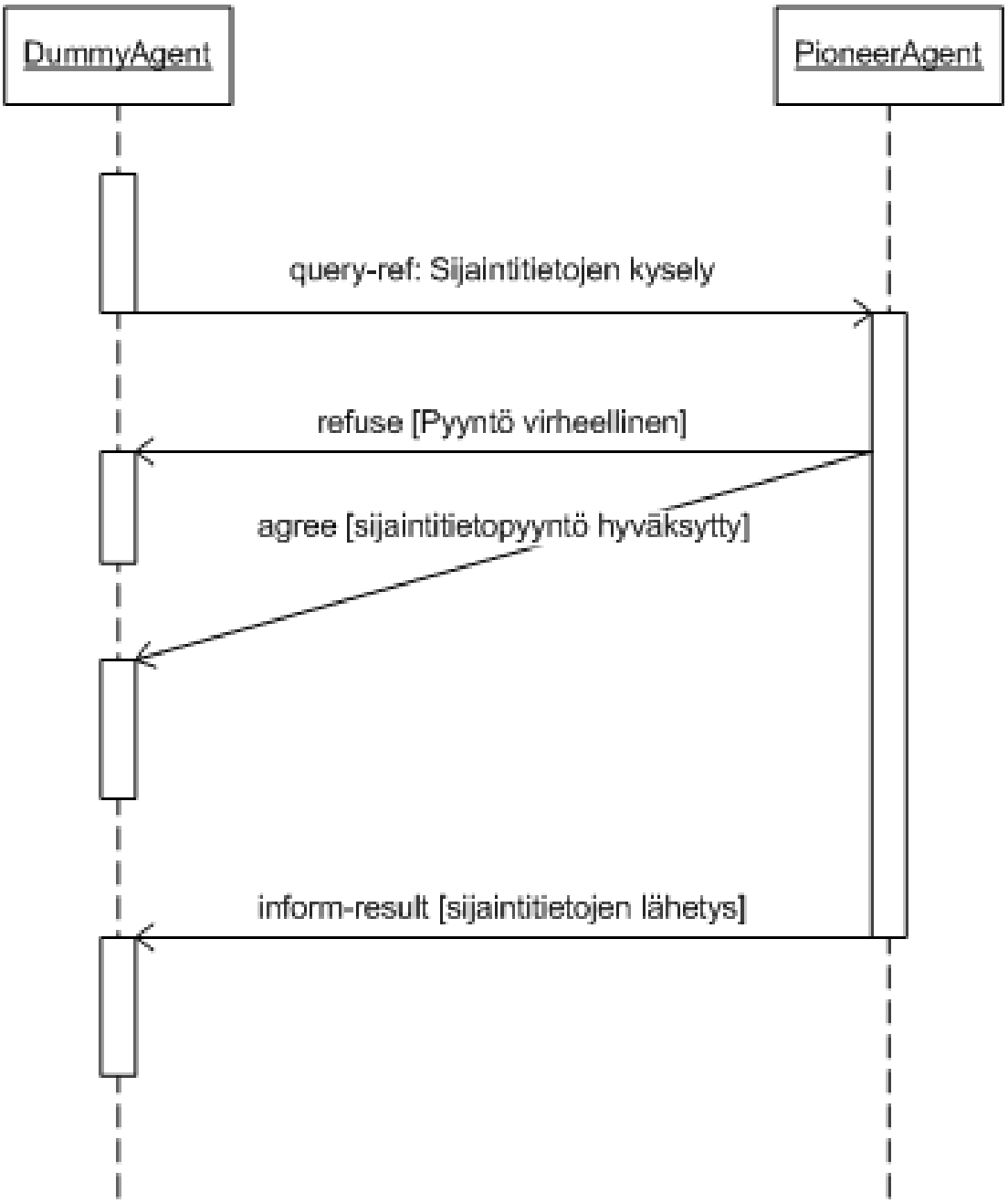
”Pioneer”-agentti hyödyntää kahta eri FIPA:n määrittelemää viestinvälitysprotokollaa. Ne ovat pyyntöprotokolla (Request-protocol) ja tiedusteluprokolla (Query-ref-protocol). Pyyntöprotokollaa käytetään paikkatietojen lähettämiseen robotille, kun taas tiedusteluprotokolla on tarkoitettu robotin paikkatietojen kyselyyn.

Pyyntöprotokolla (kuva 17) muodostuu kahdesta agentista: ”Dummy”-agentista ja ”Pioneer”-agentista. ”Dummy”-agentti on viestejä lähettävä agentti ja ”Pioneer”-agentti ottaa tässä protokollassa ainoastaan vastaan uuden paikan sijaintitiedot. Mikäli sijaintitiedot lähetetään ”request”-käskyllä virheellisesti, ”Pioneer”-agentti hylkää pyynnön. Kun ”Pioneer”-agentti hyväksyy pyynnön, se ilmoittaa ”Dummy”-agentille siirtymisestä uusien sijaintitietojen mukaiseen kohteeseen.




**Kuva 17. Pyyntöprotokolla.**

Tiedusteluprotokollan (kuva 18) avulla voidaan kysyä ”Pioneer”-agentilta robotin sijaintia. Kysyminen tapahtuu ”Dummy”-agentin avulla. Kysyminen voidaan hylätä, mikäli se ei ole protokollan mukainen. Hyväksyttyyn pyyntöön ”Pioneer”-agentti lähettää ”Dummy”-agentille vastaukseksi paikkatiedot.




**Kuva 18. Tiedusteluprotokolla.**

Molemmissa protokollissa ”Pioneer”-agentille lähetettävä viesti määritellään ”Dummy”-agentin ”content”-kenttään. Viesti pitää olla protokollan määrittelyn mukainen. Pyyntöprotokollan viesti on ”Request”-tyyppinen ja alkaa ”move”- tai ”set”-sanalla. ”Move”-sanalla alkava viesti antaa robotille komennon liikkua viestissä määriteltyyn kohteeseen. ”Set”-komento ainoastaan asettaa robotille uuden kohteen, mutta robotti ei vielä liiku sinne. Lisäksi viestiin määritellään arvot x, y ja heading. Arvot x ja y ovat pisteitä koordinaatistossa ja ”heading”-arvo on suuntakulma, johon robotti jää, kun se

pysähtyy: esim. “MOVE X:33.3 Y:33.3 HEADING 33.3” tai “SET X:33.3 Y:33.3 HEADING 33.3”.

Tiedusteluprotokollan viesti on ”Query-Ref”-tyyppinen ja se sisältää joko ”GIVE LOCATION”- tai ”GIVE ACCURACY”-viestin. ”GIVE LOCATION”-viesti palauttaa tiedon robotin sijainnista ja ”GIVE ACCURACY”-viesti palauttaa tiedon sijaintitietojen tarkkuudesta.

JADE-ohjelmistoagenttien ohjelmoimiseen tarvitaan Java-ohjelmointiin tarkoitettu ohjelmointiympäristö. Lisäksi siihen tulee määritellä tarvittavat Java- ja JADE-kirjastot. Ohjelmointiympäristössä luodaan projekti, johon itse ohjelmakoodi kirjoitetaan. Valmis ohjelmistoagentti ajetaan graafisessa JADE-kehyksessä, jossa agenttien välistä viestiliikennettä on helppo seurata. Ohjelmistoagentin luokan alkuun merkitään luokat, joita kyseinen ohjelmistoagentti tarvitsee toimiakseen. Kuvassa 19 on määritelty luokat, joita ”Pioneer”-agentti hyödyntää toiminnassaan.

```
import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.ACLMessage;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.DFService;
```

**Kuva 19. ”Pioneer”-agentin luokkamäärittely.**

”Pioneer”-agentti muodostuu itse ”Pioneer-agentti”-luokasta ja sen sisällä olevasta agentin käyttäytymistä säätelevästä sisäluokasta PioneerBehaviour. ”Pioneer”-agentti sisältää ”setup”-metodin, johon on sijoitettu agentin tunnistukseen tarvittavat tiedot.

Lopuksi ”setup”-metodi (kuva 20) sisältää ”try-catch”-lohkon, jonka sisällä on ohjelmakoodi. Koodilla luodaan ilmentymä agentin sisäluokasta PioneerBehaviour. ”PioneerBehaviour”-niminen ilmentymä asetetaan ”addBehaviour”-metodilla koko agentin käyttöön. ”PioneerBehaviour”-luokka sisältää ”action”-metodin, jossa on agentin toiminnallisuuteen liittyvä ohjelmakoodi.

```
protected void setup() {
    DFAgentDescription dfd = new DFAgentDescription();
    ServiceDescription sd = new ServiceDescription();
    sd.setType("PioneerAgent");
    sd.setName(getName());
    sd.setOwnership("Pioneer");
    sd.addOntologies("PioneerAgent");
    dfd.setName(getAID());
    dfd.addServices(sd);

    try {
        DFService.register(this,dfd);
        }
    catch (FIPAException e) {
        System.err.println(getLocalName()+" registration
        with DF unsucceeded. Reason: "+e.getMessage());
        doDelete();
         }
    try{
        PioneerBehaviour pioneerBehaviour = new
        PioneerBehaviour(this);
         addBehaviour(pioneerBehaviour);
         }
    catch(Exception e){
        System.out.println("WARNING: The agent needs the
        "+ getLocalName()+".log file.");
        e.printStackTrace();
        }
                              }}
```

**Kuva 20. ”Setup”-metodin koodi.**

”PioneerBehaviour”-luokka (kuva 21) sisältää kaiken agentin toiminnallisuuteen liittyvän. Se sisältää ”action”-metodin, jonka sisältämä koodi määrää sen, mihin agenttia käytetään. ”PioneerAgent”-agentissa ”action”-metodi sisältää robotin liikkumiseen liittyvän koodin. Agentille lähetetään viesti, joka sisältää tietyn käskyn ja se käsittelee sen ”action”-metodissa. Robotti liikkuu käskyjen mukaan. ”Action”-metodissa luodaan ensin ilmentymä msg, johon sijoitetaan agentille lähetty viesti. ”ParameterValue”-metodi osaa napata ”X:”:n, ”Y:”:n ja ”HEADING:”:in jälkeiset arvot omiin ”PioneerBehaviour”-luokan parametreihinsa. Lopuksi viesti hyväksytään ”send(reply)”-komennolla.

```
class PioneerBehaviour extends CyclicBehaviour {
    double x=0, y=0, heading=0, error_radius=0,
    error_angle=0;

    public PioneerBehaviour(Agent a) throws Exception {
        super(a);
        }

    public void action() {
        ACLMessage  msg = blockingReceive();
        ACLMessage reply = msg.createReply();
        String content = msg.getContent();
        try {
            if(msg.getPerformative()==
            ACLMessage.REQUEST) {
                if(content.startsWith("MOVE")) {
                    x = parameterValue(content,"X");
                    y = parameterValue(content,"Y");
                    heading =
                    parameterValue(content,"HEADING");
                    reply.setPerformative(ACLMessage.AGRE
                    E);
                    send(reply);
                    reply.setPerformative(ACLMessage.INFO
            RM);
                    reply.setContent("MOVED");
                    send(reply);
                      }
                }
            } catch (...) {
              ...
            }
      }
      ...
  }
```

**Kuva 21. PioneerBehaviour-luokan koodia.**

## 4.3 Pioneer-robotin testiajot

Testiajoja varten käynnistetään Pioneer-robottia ohjaava Pioneer-ohjelmistoagentti ja sille testausviestien lähettämistä varten ”Dummy”-agentti. Testauksen tarkoituksena on selvittää, toimiiko Pioneer-agenttiohjelman ja robotin rajapinnan välinen kommunikaatio toivotulla tavalla.

Testitapauksia ovat:

1. Ohjelmistoagenttien käynnistäminen.
2. X- ja Y-koordinaattien sekä ”Heading”-kulman lähettäminen Pioneer-robotille.
3. Koordinaattien kysyminen robotilta.

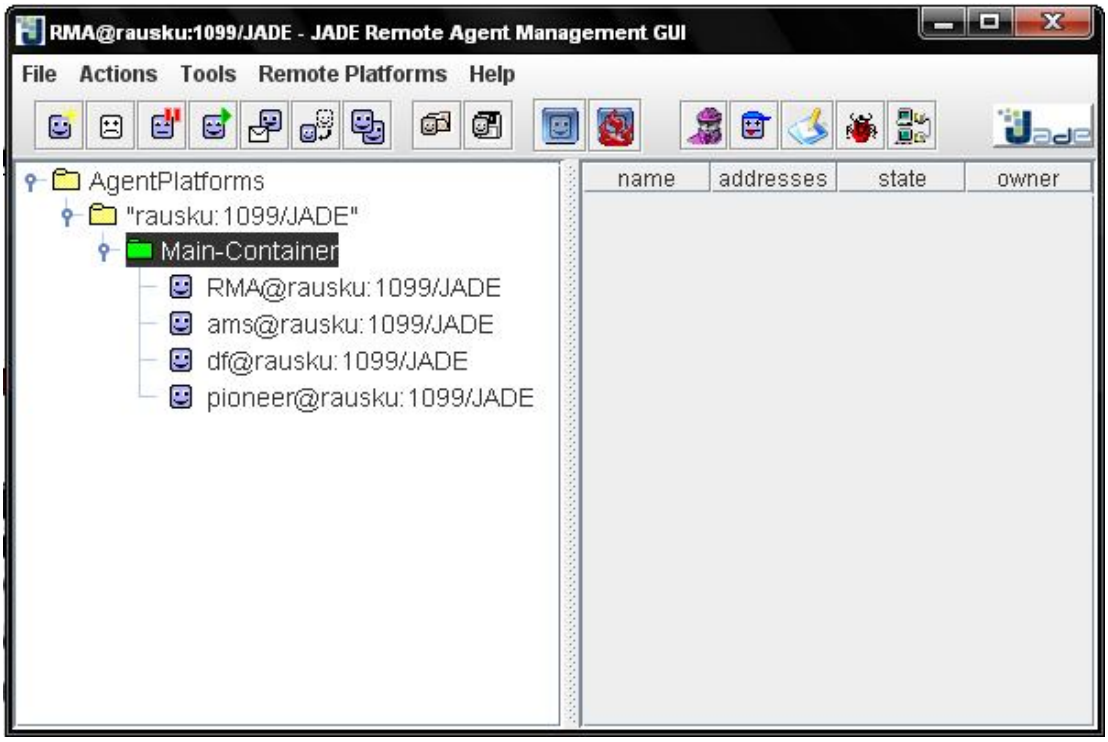
Luvussa 4.3.1 käydään läpi Pioneer-robotin käynnistäminen ja alkuvalmistelut, luvussa 4.3.2 ”Heading”-kulman testaus, luvussa 4.3.3 X- sekä Y-koordinaattien lähettämisen testaus ja luvussa 4.3.4 robotin sijaintikyselyn testaus.

### 4.3.1 Pioneer-robotin käynnistäminen ja alkuvalmistelut

Ohjelmistoagenttiympäristö käynnistetään komennolla ”java jade.Boot –qui”. Komennon jälkeen avautuu JADE-ikkuna, jolla eri ohjelmistoagenttien luominen graafisen ikkunan kautta on mahdollista. Komento käynnistää myös robotin, joka alkaa pitää naksuttavaa ääntä. Jotta robotille voitaisiin lähettää viestejä, täytyy käynnistää ”Pioneer”-agentti, jonka tehtävänä on kommunikoida robotin ARIA-rajapinnan kanssa. ”Class Name”-kohtaan valitaan ”Pioneer”-agentti listasta, josta tässä testitapauksessa löytyy yhteensä 11 luokkaa. Agentti nimetään, tässä tapauksessa nimellä Pioneer (kuva 22).

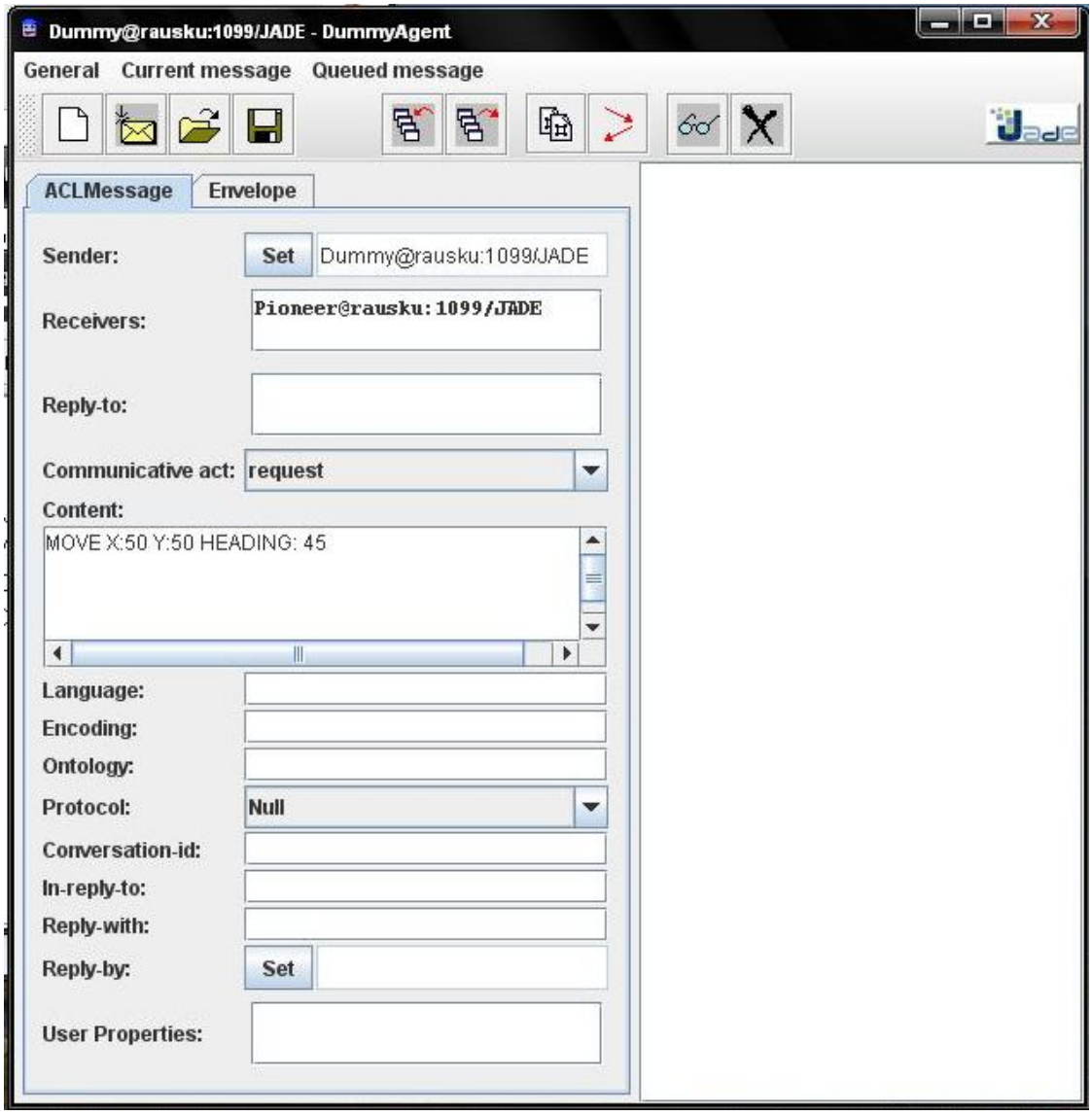

**Kuva 22. Pioneer-agentin nimeäminen.**

”Pioneer”-agentti tunnistetaan jatkossa JADE-kehyksessä nimen perusteella. Kun tiedot on täytetty ja hyväksytty, agentti ilmestyy näkyville JADE-kehykseen, joka näkyy kuvassa 23.




**Kuva 23. JADE:n graafinen käyttöliittymä.**

Seuraavaksi luodaan ”Dummy”-agentti samalla tavalla kuin ”Pioneer”-agentti. Listasta valitaan dummyagent ja viestin vastaanottajaksi valitaan ”Pioneer”-agentti. ”Communicative Act”-valikosta valitaan viestin tyypiksi Request. Liikkumiskomennot kirjataan ”Dummy”-agentin ”Content”-kenttään muodossa ”MOVE X:_ Y:_ HEADING:_”. Viestin lähettämisen jälkeen robotti liikkuu koordinaatiston pisteisiin X ja Y. Robotti jää ”heading”-luvun osoittamaan kulmaan. Sekä koordinaatit että kulma voivat olla desimaalilukuja, mutta silloin desimaaliluvun pilkun tilalla on käytettävä pistettä tai muussa tapauksessa ”Dummy”-agentti antaa "not understood"-virheilmoituksen. Kuvassa 24 on esimerkki ”Dummy”-agentin viestinlähetysikkunasta ja itse viestistä.




**Kuva 24. Dummy-agentti.**

### 4.3.2 ”Heading”-kulman testaus

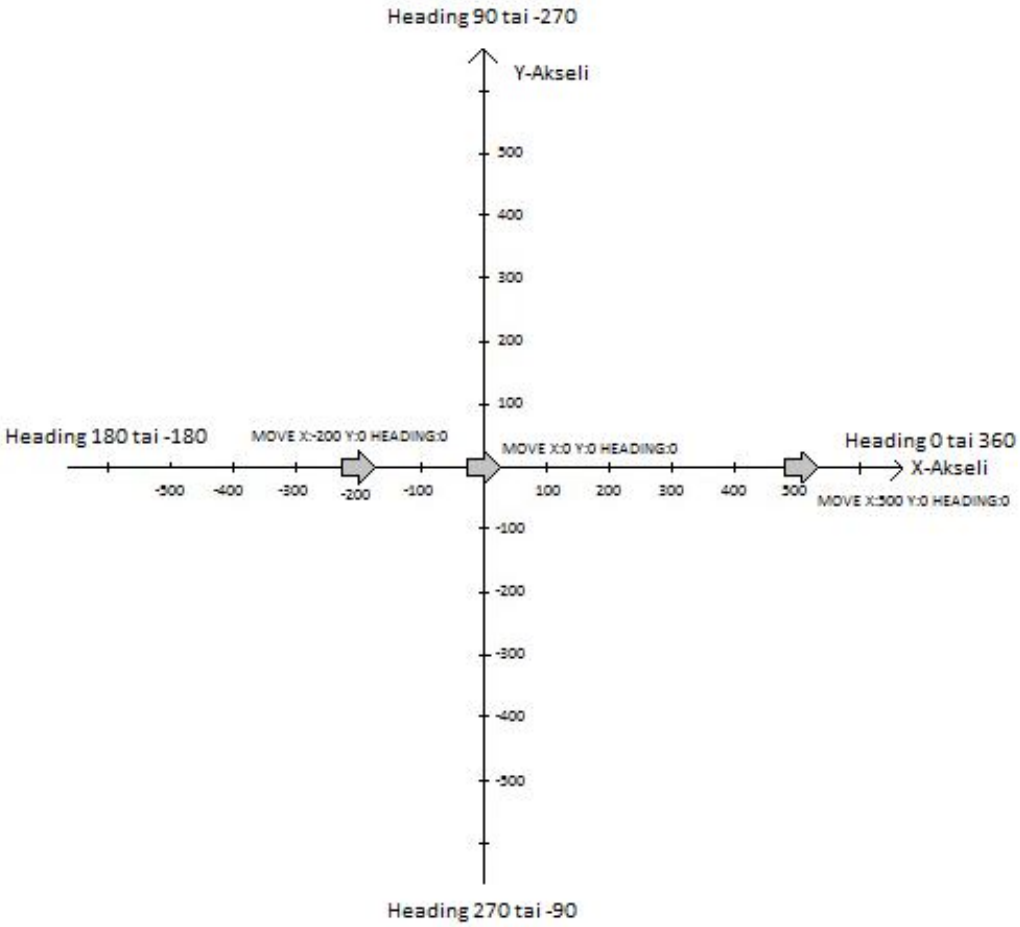
”Heading”-kulma on robotin asentokulma, johon robotti jää, kun se pysähtyy. Kulman testaamista varten annettiin aluksi X- ja Y-koordinaattien arvoksi nolla sekä muutettiin ”heading”-kulmaksi 360 astetta. Viesti oli tällöin muotoa ”X:0 Y:0 HEADING:360” ja robotti pyörähti paikallaan 360 astetta. Kun edelleen vaihdettiin ”heading”-kulmaksi 90 astetta, robotti kääntyi 90 astetta vastapäivään. Käskystä ”heading 0”, robotti kääntyi taas takaisin alkukulmaansa. Robotin kääntymisliikkeet tapahtuivat kiinteän koordinaatiston mukaisesti, jolloin robotin käytön alkuarvoina on ”X:0 Y:0 HEADING:0”. Mii-

nusmerkkinen ”heading”-arvo sai robotin kääntymään myötäpäivään. Robotti kääntyi testattuihin ”heading”-kulmiin ilman havaittuja virheitä.

### 4.3.3 X- ja Y-koordinaattien lähettämisen testaus

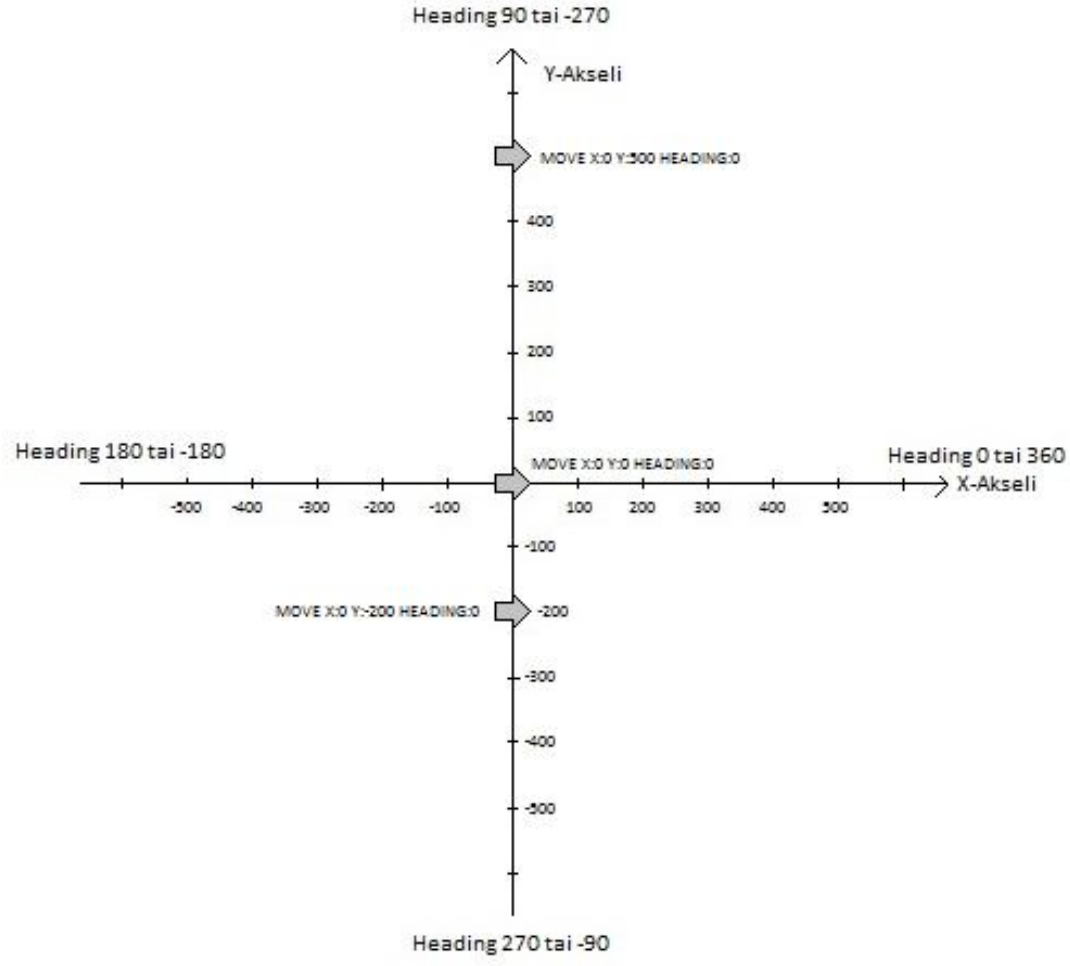
X- sekä Y-koordinaatteja testattiin antamalla ensin pelkälle X-koordinaatille erilaisia arvoja ja Y:n arvo pidettiin nollana. Sen jälkeen vastaavasti testattiin Y-koordinaatteja erilaisilla arvoilla X:n oltua nolla. Lopuksi kokeiltiin muuttaa X- ja Y-koordinaatteja samanaikaisesti. Myös eri ”Heading”-kulmia testattiin samalla, jotta saatiin varmistettua, ettei sen muutoksista aiheudu häiriötä robotin liikkumiselle.

Ensin käynnistettiin ”Pioneer”-agentti ja seuraavaksi ”Dummy”-ohjelmistoagentti, jolla lähetettiin robotille halutut koordinaatit ja kulma. Robotti liikkui näitä tietoja hyödyntäen haluttuun koordinaatiston pisteeseen. Testin lähtöpisteenä käytettiin origoa, johon robotti liikkui komennolla ”MOVE X:0 Y:0 HEADING:0”. ”Dummy”-agentin viestistä ”MOVE X:500 Y:0 HEADING:0”, robotti kulki suoraan eteenpäin 50 cm. Muuttamalla arvoiksi ”MOVE X:0 Y:0 HEADING:0”, robotti saatiin kääntymään ympäri ja palaamaan lähtöpisteeseensä. Koordinaateilla ”MOVE X:-200 Y:0 HEADING:0”, robotti kääntyi myötäpäivään 180 astetta ja liikkui 20 cm. Lopuksi se kääntyi vastapäivään 180 astetta takaisin lähtökulmaan. ”Heading”-kulmaksi ja X- sekä Y-koordinaateiksi kelpasivat myös desimaaliluvut, kunhan pilkun tilalla käytettiin pistettä. Kyseisiä testitapauksia selventää kuva 25, jossa koordinaatistoon on merkitty robotin sijainti ja suunta testauksen eri vaiheissa. Nuolen suunta kuvaa robotin ”heading”-kulmaa, johon robotti kääntyy liikkumisen lopuksi.



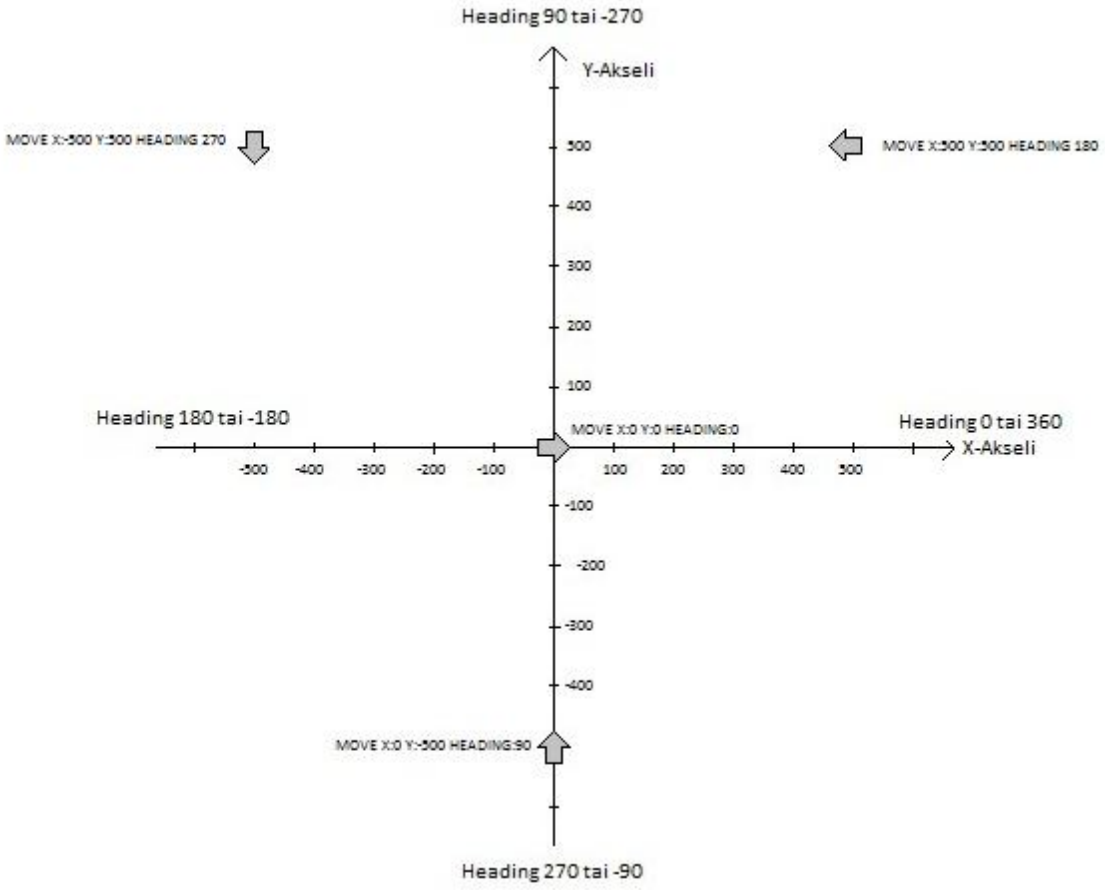

**Kuva 25. X-koordinaatin testaus.**

Myös Y-koordinaatteja testattiin samalla tavalla. ”MOVE X:0 Y:500 HEADING:0”-komennolla robotti kääntyi 90 astetta vastapäivään ja liikkui 50 cm koordinaatiston y-akselin suuntaisesti. Oletuksena oli, että testissä robotin lähtöpiste on origo. ”MOVE X:0 Y:-200 HEADING:0”-komennolla robotti kääntyi 90 astetta myötäpäivään ja liikkui 20 cm eteenpäin. Molemmissa tapauksissa robotti kääntyi lopuksi lähtökulmaansa heading 0 astetta. Testitapaukset on piirretty koordinaatistoon, joka näkyy kuvassa 26. Nuolen suunta vastaa robotin ”heading”-kulmaa.




**Kuva 26. Y-koordinaatin testaus.**

Koordinaattien ja kulman testausta jatkettiin kaikkien kolmen tiedon (X- sekä Y-koordinaattien ja heading-kulman) syöttämisellä samanaikaisesti. Lähtöpisteenä oli koordinaatiston origo (0,0), johon robotti liikkui aluksi komennolla ”MOVE X:0 Y:0 HEADING:0”. Ensimmäisenä annettiin komento ”MOVE X:500 Y:500 HEADING:180”. Robotti kääntyi tuolloin noin 45 astetta vastapäivään ja eteni haluttuun pisteeseen sekä kääntyi sitten 135 astetta vastapäivään haluttuun kulmaan (180 astetta). Toisena testikomentona oli ”MOVE X:0 Y:-500 HEADING:90”. Tällä komennolla robotti eteni suoraan esitettyyn koordinaatiston pisteeseen ja kääntyi lopuksi myötäpäivään kulmaan 90 astetta. Seuraavaksi annettiin komento ”MOVE X:-500 Y:500 HEADING: 270”. Tuolloin robotti liikkui haluttuun pisteeseen ja kääntyi kulmaan 270 astetta. Kuvassa 27 on koordinaatistossa kaikkien kolmen testipisteen sijainti koordinaatistossa. Nuolen suunta kuvaa robotin ”heading”-kulmaa.



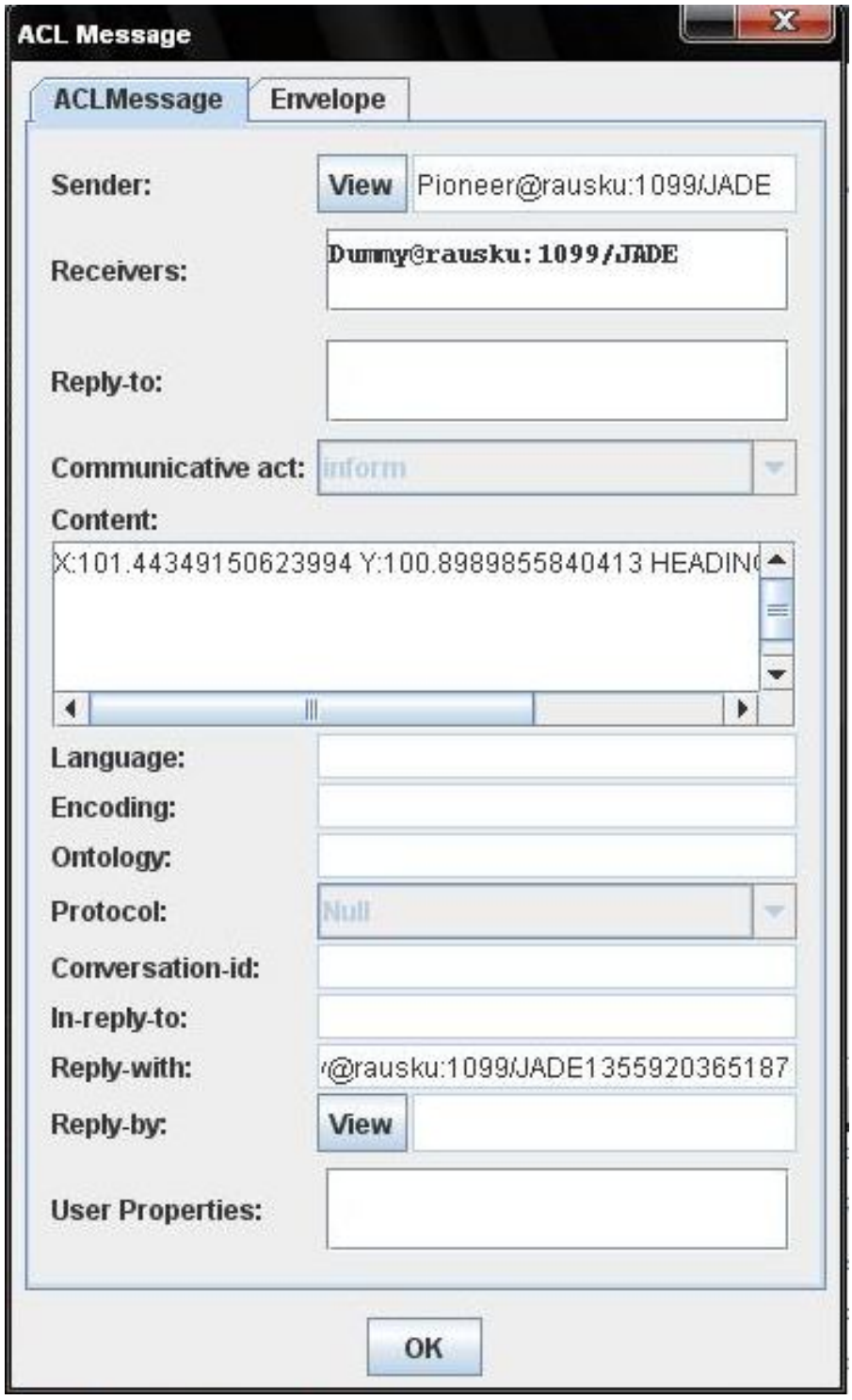

**Kuva 27. X- ja Y-koordinaattien sekä ”heading”-kulman testaus.**

Kaikki X- ja Y-koordinaattien testitapaukset toimivat ilman havaittuja virheitä. Robotin akku oli kuitenkin niin huono, että robottia täytyi pitää kiinni verkkovirrassa koko testauksen ajan.

### 4.3.4 Robotin sijaintikyselyn testaus

Robotilta voi kysyä ”GIVE LOCATION”-komennolla paikkatietoja. Komento kirjoitetaan ”Dummy”-agentin ”content”-kenttään. Robotti vastaa tiedon, mutta viestin saa näkymään ainoastaan ”Sniffer”-agentilla katsomalla viestin graafisessa JADE-kehyksessä. Tarkempaa tietoa ”sniffer”-agentin käytöstä löytyy luvusta 2.3. Arvot ovat tarkkoja useiden kymmenien desimaalien arvoja, joissa näkyy robotin tarkka sijainti koordinaatistossa. Aina ei päästä aivan haluttuun pisteeseen, mutta virhemarginaalit ovat erittäin

pieniä, kokoluokaltaan yhden millimetrin kymmenes- tai sadasosia. Kuvassa 28 on esimerkki ”GIVE LOCATION”-komennon palautusviestistä. Robotin tarkka arvo näkyy kuvan ”content”-kentässä. Robotille lähetettiin komento ”MOVE X:100.9 Y:100.9 HEADING:0”. Kuvasta 28 näkyy, että robotti ei ole aivan tarkalleen annetussa pisteessä.




**Kuva 28. ”GIVE LOCATION”-komennon vastausviesti.**

## 4.4 Yhteenveto

Tavoitteena oli käyttää ohjelmistoagentteja Pioneer 3DX -robotin ohjaukseen. Tämä tutkielma perehtyi ainoastaan alemman tason agenttiin, jonka tehtävänä on välittää koordinaatit robotille. Lisäksi robotilta on mahdollista kysyä sen sijaintikoordinaatit. ”Pioneer”-agentin ohjelmoinnissa määritellään ensin agentin tunnistetiedot, joiden avulla mahdollistetaan agenttien välinen viestiminen. Sille täytyy luoda oma käyttäytyminen omalla ”PioneerBehaviour”-luokalla. ”Pioneer”-agentti hyödyntää kahta FIPA:n määrittelemää viestinvälitysprotokollaa, joita ovat pyyntöprotokolla (Request-protocol) ja tiedusteluprotokolla (Query-ref -protocol). Tiedusteluprotokollalla kysytään robotin paikkatietoja ja pyyntöprotokollalla lähetetään paikkatiedot robotille.

Ohjelmistoagenttien hallinta tapahtuu JADE-ohjelmistoagentti ympäristössä, jossa on agenttien hallintaan oma graafinen sovelluksensa. Ohjelmistoagenttien ohjelmointiin täytyy määritellä tarvittavat JADE- sekä Java-kirjastot ja ohjelmoinnin voi tehdä millä tahansa Java-ohjelmointiympäristöllä.

Testiajoissa testattiin sekä robotin että ohjelmistoympäristön käynnistykseen liittyvät asiat, koordinaattien syöttö, robotin liikkuminen ja robotin sijainnin ilmoittamiseen liittyvät seikat. Ohjelmistoympäristön käynnistäminen tapahtui komennolla "java jade.Boot -gui" ja JADE:n graafinen ohjelmistoagenttikehys käynnistyi ongelmitta. Lisäksi luotiin Pioneer-robottia ohjaava ohjelmistoagentti ja sille viestejä lähettävä ”Dummy”-agentti ilman havaittuja virheitä. Koordinaattien tai ”heading”-kulman lähettämisessä ja robotin liikkumisessa ei havaittu merkittäviä virheitä. Pientä laskuvirhettä robotin liikkumisessa annettuun pisteeseen oli havaittavissa, kun katsottiin ”GIVE LOCATION”-komennolla robotin täysin tarkka sijainti.

# 5 POHDINTA

Robottien lukumäärä kasvaa jatkuvasti. Niiden yleistymiseen on monia syitä, joista suurimpana on niiden kehittyminen helpommin ohjelmoitavaksi erilaisiin käyttökohteisiin. Robottien yleistyminen erityisesti teollisuudessa on laskenut niiden hankintahintoja. Monet teollisuuden tuotteet ovat niin kehittyneitä ja valmistukseltaan tarkkuutta vaativia, että niiden kokoaminen erittäin suurta tarkkuutta ja nopeutta hyödyntävällä robotilla on ainoa mahdollisuus saavuttaa tarpeeksi suuret tuotantomäärät.

Uskon, että teollisuudessa roboteilla yritetään jatkossa tehdä aiempaa integroidumpia järjestelmiä, jotta tulevaisuudessa tehtaat voisivat toimia lähes itsenäisesti. Tehdaslinjaston alussa syötettäisiin tällöin materiaali ja lopusta tulisi valmis tuote. Robotit ovat nykyään kokoamistöissä jo niin nopeita, etten usko niiden nopeuden enää jatkossa merkittävästi kasvavan. Ennemminkin sen sijaan tulevaisuudessa robottien muunneltavuutta voidaan kehittää niin, että samaa robottia voidaan ohjelmamuutoksen avulla käyttää eri käyttökohteisiin.

Tulevaisuudessa on mahdollista myös erilaisten uusien käyttökohteiden hyödyntäminen robotiikan avulla. Roboteista voidaan tehdä kooltaan erittäin pieniä esimerkiksi lääketieteelliseen käyttöön, jossa niiden avulla voidaan korjata soluja ja avata verisuonia. Tämän tieteenlajin kehitys on tietysti suotavaa, mutta samalla se herättää kysymyksiä siitä, miten saadaan hallittua näitä erittäin pieniä nanorobotteja turvallisesti.

Yksi mielenkiintoinen ja varmasti tulevaisuudessa eniten kehittyvä robotiikan ala on palvelurobotiikka. Palvelurobotiikan tehtävänä on palvella ja helpottaa ihmisen työtä. Robotiikka on ollut perinteisesti enemmän teollisuuden suosima ala, mutta nyt se on tulossa myös koteihin erilaisten ihmisen toimintaa helpottavien laitteiden avulla. Tulevaisuudessa kodeissa voi olla esimerkiksi androideja, joita voidaan opettaa tekemään erilaisia kodin askareita. Tuolloin ihminen voisi unohtaa ainakin siivoamisen ja pyykinpesun.

Tässä tutkielmassa perehdyttiin olennaisena osana JADE-ohjelmistoagentteihin. JADE-ohjelmistoagentit ovat pieniä itsenäisesti toimivia ohjelmia, jotka kommunikoivat keskenään. Niille on määritelty tarkat standardit ja ne helpottavat ohjelmistoagenttien integrointia yhteen. Ohjelmistoagenteille on luotu omat arkkitehtuurimallit, joita hyödyn-

tämällä saadaan helpotettua järjestelmien kehitystä ja tuotua ohjelmiston rakenteeseen sekä toimintaan selkeyttä.

Ohjelmistoagenttien hyödyntäminen osana robottien kehitystä on mahdollista. Tässä tutkielmassa ohjelmistoagentteja käytettiin viestien välittäjänä Pioneer-robotille. Idea itsessään oli mielenkiintoinen ja toimiva. Konkreettisemmaksi robotin ohjaaminen kuitenkin muuttuisi vasta sitten, kun agentit saisivat robotin ohjaukseen tarvittavan tiedon joltakin itsenäisesti toimivalta anturilta. Näin ollen robotti voisi olla sekä itsenäisesti että agenteille lähetettävillä viesteillä ohjautuva.

Ohjelmistoagentin melko yksinkertaisesta perusrakenteesta johtuen yksinkertainen ohjelmistoagentti ei tarvitse kovin suurta ohjelmointiosaamista. Sen sijaan robotin liikkeiden hallintaan perustuva, FT Mauno Rönkön ohjelmoima, C++-kielellä toteutettu robotin oman rajapinnan ja Pioneer-ohjelmistoagenttiohjelman välissä oleva ohjelma vaatii jo enemmän osaamista.

JADE-ohjelmistoagenttien luominen ja seuraaminen on tehty helpoksi, jos käyttää JADE:n omaa graafista ohjelmistoagenttisovellusta. Sillä pystyy luomaan agentteja ja lähettämään niille viestejä. Myös ohjelmistoagenttien viestien seuranta on tällöin vaivatonta. Ilman tätä mahdollisuutta agenttien välistä viestittelyä olisikin vaikea seurata.

Tämän tutkielman testijärjestelyissä testattiin JADE-ympäristön soveltumista robotin ohjaamiseen. Ohjelmistoagenttiympäristö toimi erittäin hyvin. Viestien lähettäminen robotille onnistui ongelmitta ja robotti liikkui täysin koordinaattien mukaan. Testauksen aikana ei havaittu virheitä.

Mikäli Pioneer-robotinohjausta haluttaisiin kehittää, täytyisi luoda jokin järjestelmä, joka osaisi generoida ohjausviestejä automaattisesti, kun esimerkiksi jokin este tulee eteen. Tällä nykyisellä tasolla järjestelmä on vielä melko yksinkertainen ja sitä ei voi käyttää kuin ainoastaan robotin liikuttamiseen avoimella alustalla, jossa esteet ovat helposti havaittavissa. Käytännön sovelluskohteet ovatkin melko vähissä nykyisen kaltaiselle ratkaisulle. Toivoisin, että joku ottaisi vielä tehtäväkseen idean kehittämisen. Pioneer-robottia hyödynnetään nykyään lähinnä tutkimuskäytössä. Tulevaisuudessa muutamia käyttökohteita kyseiselle robotille reaalimaailmassa voi kuitenkin ilmaantua. Robottia voisi nimittäin käyttää esimerkiksi tavaroiden kuljetukseen. Toisaalta, jos robot-

tiin kiinnitettäisiin videokamera ja valonlähde, sitä voitaisiin käyttää myös erilaisten ahtaiden paikkojen tutkimiseen.

Robottien käyttömahdollisuudet ovat nykyisinkin jo varsin laajat ja jatkossa niitä tullaan hyödyntämään entistä monipuolisemmin. Tällä hetkellä robotit ovat eritäin yleisesti käytössä teollisuudessa, mutta erityisesti yksityiselämässä niille voisi olla lukuisia käyttökohteita. Tulevaisuudessa onkin mahdollista automatisoida monia eri arjen askareita. Kaupat voisivat olla suuria robottiautomaatteja, joille syötettäisiin ostoslistan mukainen tilaus ja robotti toimittaisi tuotteet varastosta. Lisäksi autot voisivat liikkua robotin ohjaamana ja myös koiran ulkoiluttamisen sekä lumitöiden tekemisen voisi hoitaa robotti. Tulevaisuudessa robottien ja niiden sovellusten käyttömahdollisuudet ovatkin lähes rajattomat.

# LÄHTEET


[Ade12] Adept Technology Inc.: *Pioneer P3-DX Datasheet.* Mobilerobots Inc., 2012. http://www.mobilerobots.com/Libraries/Downloads/Pioneer3DX-P3DX-RevA.sflb.ashx (12.2.2013)

[BCG07] Bellifemine F., Caire G., and Greenwood D.: *Developing Multi-Agent Systems with JADE*. John Wiley & Sons, Ltd, 2007.

[BCP03] Bellifemine, F., Claire, G., Poggi, A., and Rimassa, G.: JADE – A White Paper. *Exp*, Vol. 3, No. 3, September 2003. http://sharon.cselt.it/projects/jade/papers/WhitePaperJADEEXP.pdf (12.2.2013)

[Enq99] Enqvist K.: Ihmisyys nanotekniikan peilissä. *Duodecim,* Vol. 23, pp. 2545–2550, 1999.

[FIP13] FIPA: *The Foundation for Intelligent Physical Agents.* FIPA, 2013. http://www.fipa.org (12.2.2013)

[Fou02] Foundation for intelligent Physical Agents: *FIPA Request Interaction Protocol Specification.* Foundation for Intelligent Physical Agents, SC00026H, 2002. http://www.fipa.org/specs/fipa00026/SC00026H.pdf (12.2.2013)

[Fou04] Foundation for Intelligent Physical Agents: *FIPA Agent Management Specification.* Foundation for Intelligent Physical Agents, SC00023K, 2004. http://www.fipa.org/specs/fipa00023/SC00023K.pdf (12.2.2013)

[HoK96] Howard A. and Kichen L.: Generating Sonar Maps in Highly Specular Environments. *Proceeding of the Fourth International Conference on Control Automation Robotics and Vision,* 1996, pp. 1870–1874.

[HSB03] Hähnel D., Schulz D., and Burgard W.: Mobile Robot Mapping in Populated Environments. *Advanced Robotics*, Vol. 17, No. 7, pp.579–598, 2003.

[JAT10] Jade Administration Turorial: *Turorial 1 – JADE Architecture Overview.* Jade Administration Turorial, 2010. http://jade.tilab.com/doc/tutorials/JADEAdmin/jadeArchitecture.html (12.2.2013)

[Lah08] Lahden Ammattikorkeakoulu: *Robotiikka.* Verkkodokumentti, Lahden Ammattikorkeakoulu. http://miniweb.lpt.fi/automaatio/opetus/luennot/pdf_tiedostot/Robotiikka_yleinen.pdf (12.2.2013)

[Mob07] MobileRobots Inc.: *Pioneer 3 Operations Manual.* MobileRobots Inc., 2007. https://docs.google.com/viewer?a=v&q=cache:ty6vCdScjIkJ:143.106.50.145:8080/Cursos/IA368N/02-09/P3OpMan5.pdf+Pioneer+3+Operations+Manual,+2007&hl=fi&gl=fi&pid=bl&srcid=ADGEEShgffXPXrptv90SuNxTCf4HR1A9_UA_pDkpycGMhOnsgOcqwWI2vZeqKYYGnLUdTR6mES4ianG4LttbulQl6SJMLHVJlAvbngRwI8MnUYLNrY9G1-tksPEWT_o7I_19cvbH08C3&sig=AHIEtbR_dAa0s2t9Ob7XhHU4yHOOqwgsxQ (12.2.2013)

[New06] NewScientist: *Robot Moved by a Slime Mould's Fears.* NewScientist, 2006. http://www.newscientist.com/article/dn8718 (12.2.2013)

[Nii09] Niininen J.: *Ohjelmistoagentit robottiarkkitehtuurissa.* Pro Gradu -tutkielma, Tietojenkäsittelytieteen laitos, Kuopion yliopisto, 2009.

[Rob99] Suomen Robotiikkayhdistys: *Robotiikka.* Suomen Robotiikkayhdistys, 1999.

[Tha97] Thau R.: *Reliability Mapping a Robot's Enviroment Using Fast Vision and Local, But not Global, Metric Data.* Ph.D. Thesis, Massachusetts Institute of Technology, 1997.

[Thr02] Thrun S.: *Robotic Mapping – A Survey.* School of Computer Science, Carnegie Mellon University, Pittsburgh, USA, February 2002. http://reports-archive.adm.cs.cmu.edu/anon/2002/CMU-CS-02-111.ps (12.2.2013)

[Wiki13] Wikipedia: *ASIMO.* Wikipedia, 2013. http://fi.wikipedia.org/wiki/ASIMO (12.2.2013)

[Fou02a] Foundation for intelligent Physical Agents: *FIPA Query Interaction Protocol Specification.* Foundation for Intelligent Physical Agents, SC00027H, 2002. Viitattu 26.3.2010, saatavilla osoitteesta: http://www.fipa.org/specs/fipa00027/SC00027H.pdf