FIPA-agenttiarkkitehtuuriin pohjautuvat rationaaliset agentit

Janne Järvi

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos
Tietojenkäsittelyoppi
Pro gradu -tutkielma
Lokakuu 2003

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos
Janne Järvi: FIPA-agenttiarkkitehtuuriin pohjautuvat rationaaliset agentit
Pro gradu -tutkielma, 74 + 4 sivua
Lokakuu 2003

Agentteja voidaan toteuttaa monella tavalla. Eräs tapa perustuu perustuu BDI-teoriaan, jonka mukaan älykkään agentin, kuten ihmisen, toiminta perustuu uskomuksiin (belief), toiveisiin (desire) ja aikomuksiin (intention). Tämä intentionaalinen tulkinta järjestelmästä tuo uuden abstraktiotason ohjelmointiin.

Moniagenttijärjestelmissä agentit kommunikoivat lähettämällä viestejä toisilleen. Nämä keskustelut noudattavat erityisiä interaktioprotokollia, jotta agentit kykenisivät koordinoimaan päättelynsä ja välttämään lukkiutumisen. Yleensä monen agentin järjestelmässä on yksi tai useampi agentti tai muu komponentti, joka pitää kirjaa järjestelmän agenteista ja niiden tarjoamista palveluista. FIPA (Foundation for Intelligent Physical Agents) on yhteisö, jossa määritellään tällaisten järjestelmien vaatimia palveluita, viestikieliä sekä interaktioprotokollia.

Esittelen tässä tutkielmassa erään FIPA-pohjaisen agenttiympäristön ja siihen suunnittelemani ja toteuttamani BDI-agenttiarkkitehtuurin. Toteutus ei sovellu vielä monimutkaisten järjestelmien toteuttamiseen, mutta sen avulla voi toteuttaa nopeasti erilaisia prototyyppejä.

Avainsanat ja -sanonnat: ohjelmistoagentit , moniagenttijärjestelmät, BDI-malli, FIPA.

Sisällys

1 Johdanto

Agentilla tarkoitetaan ohjelmakomponenttia, joka tekee havaintoja jossakin ympäristössä sekä suorittaa toimintoja siinä saavuttaakseen omat tai sille annetut päämäärät. Agentti reagoi ympäristössä tapahtuviin muutoksiin. Agentti pyrkii myös ennakoimaan ympäristön muutoksia, eli toimimaan proaktiivisesti. Agenttitutkimus on hieman laantunut 90-luvun puolivälin kovan huuman jälkeen. Parhaimmillaan agenttipohjaista ohjelmistokehitystä pidettiin seuraavana ohjelmistokehityksen läpimurtona ja jopa ohjelmoinnin vallankumouksena. 2000-luvulle tultaessa suhtautuminen (osittain alkaneen taloudellisen laskusuhdanteen saattelemana) on muuttunut realistisemmaksi. Agenttiteknologia nähdään vain yhdeksi tavaksi tehdä ohjelmistoja.

Rationaalisia agentteja voidaan toteuttaa monella tavalla. Eräs tapa perustuu M.E. Bratmanin [1987, 1991] kehittämään BDI-teoriaan, jonka mukaan älykkään agentin, kuten ihmisen, toiminta perustuu uskomuksiin (belief), toiveisiin (desire) ja aikomuksiin (intention). Agentin uskomukset ovat faktoja ympäristöstä, joiden agentti uskoo pitävän. Toiveet ovat asiantiloja, jotka agentti haluaisi ideaalitilanteessa saavuttaa, ne voivat olla keskenään ristiriitaisia. Agentin aikomukset ovat toiveista suodatettuja tulevaisuuteen suunnattuja asiantiloja, joiden perusteella agentti voi suunnitella tulevaa toimintaansa sekä koordinoida toimintaansa muiden agenttien suhteen. Agentin aikomukset eivät voi olla keskenään ristiriitaisia.

Moniagenttijärjestelmät (Multiagent Systems, MAS) ovat tärkeä osa-alue hajautetun tekoälyn tutkimuksessa. Niissä agentit tekevät yhteistyötä saavuttaakseen yhdessä järjestelmälle asetetut kokonaistavoitteet. Tyypillisesti moniagenttijärjetelmää käytetään ongelmissa, joita on vaikea ratkoa keskitetysti. Monen agentin järjestelmissä agentit kommunikoivat lähettämällä viestejä toisilleen. Nämä keskustelut noudattavat erityisiä interaktioprotokollia, jotta agentit kykenisivät koordinoimaan päättelynsä ja välttämään lukkiutumisen. Yleensä monen agentin järjestelmässä on yksi tai useampi agentti tai muu komponentti, joka pitää kirjaa järjestelmän agenteista ja niiden tarjoamista palveluista. FIPA (Foundation for Intelligent Physical Agents) on yhteisö, jossa määritellään tällaisten järjestelmien vaatimia palveluita, viestikieliä sekä interaktioprotokollia. FIPA-agentit kommunikoivat keskenään käyttämällä FIPA ACL -viestikieltä. Tämän kielen semantiikka on määritelty BDI-teorian primitiiveillä, joten BDI-arkkitehtuuri sopii hyvin FIPA-agentin toteutustavaksi.

Tutkielman loppuosa rakentuu seuraavasti: Luvussa kaksi esitän joitakin esimerkkejä agentin määritelmistä sekä luon katsauksen agenttiohjelmoinnin historiaan. Luvussa kolme käyn läpi erilaisia teorioita, joita käytetään

agenttitutkimuksessa. Luvussa neljä esittelen toteutettuja agenttiarkkitehtuureja. Luvussa viisi esittelen moniagenttijärjestelmän keskeisiä ominaisuuksia. Luvussa kuusi esittelen FIPA-yhteensopivan agenttijärjestelmän sekä järjestelmässä toimivan BDI-agentin arkkitehtuurin. Luvussa seitsemän esittelen FIPA-ympäristöön toteuttamaani BDI-agenttia ja pohdin mahdollisia kehityssuuntia.

2 Ohjelmistoagentit

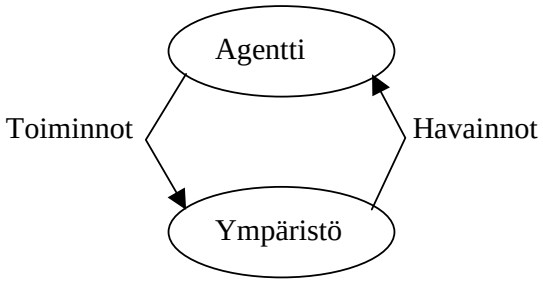
Kuten monen muun vakiintumattoman käsitteen, myös agentin määritelmistä esitetään aina vastalauseita. Niiden mukaan jotakin osa-aluetta on painotettu liikaa tai liian vähän, tai jotain tärkeätä on jätetty kokonaan pois [Jennings, 2000]. Tässä luvussa esittelen joitakin agentin määritelmiä sekä agenttiohjelmoinnin historiaa ja joitakin käsitteitä.

2.1 Agentin määritelmiä

Agentin määrittely on osoittautunut vaikeaksi tehtäväksi. Tutkijoiden parissa ei näytä vallitsevan täydellistä yksimielisyyttä agentin olemuksesta. Alan kirjallisuudessa esitetään kuitenkin usein Russelin ja Norvigin [1995] sekä Wooldridgen [1995, 1999] esittämät määritelmät.

2.1.1 Russel ja Norvig: Agentit toimivat ympäristössä

Russelin ja Norvigin [1995] mukaan agentiksi voidaan määritellä mikä tahansa kokonaisuus, joka havainnoi ja toimii jossakin ympäristössä. Ihmisagentti tekee havaintoja silmillä ja korvilla, ja toimii käsillä, jaloilla ja suulla. Robotti havainnoi kameralla, infrapunalla tai muulla vastaavalla laitteella, ja suorittaa toimintoja erilaisilla moottoreilla. Ohjelmistoagentilla havainnot ja toiminnot ovat erilaisia bittijonoja.



Kuva 2.1 Agentti toimii ja havainnoi ympäristössä.

Agentin ympäristön ominaisuudet vaikuttavat agentin päätöksentekoprosessin monimutkaisuuteen. Agentin ympäristö voidaan jakaa ominaisuuksiensa perusteella viiteen eri tyyppiin [Russel and Norvig, 1995] :

- *Avoin ja suljettu ympäristö.* Avoimessa ympäristössä agentin ympäristöstä saama tieto on täydellistä ja ajantasalla olevaa. Yleensä agentin ympäristö on suljettu. Se ei saa täydellistä tietoa ympäristöstään. Mitä

avoimempi ympäristö on, sitä yksinkertaisempaa agentin ohjelmoiminen on.

- *Deterministinen ja epädeterministinen ympäristö.* Deterministisessä ympäristössä jokaisella agentin toiminnolla on yksi taattu vaikutus. Epädeterministisessä ympäristössä agentin toiminnon lopputulosta ei voida tietää etukäteen. Esimerkiksi meitä kaikkia ympäröivä fyysinen maailma voidaan luokitella epädeterministiseksi ympäristöksi. Agentin ohjelmoiminen toimimaan epädeterministisessä ympäristössä on haasteellisempaa kuin deterministisessä, koska sen päättelyssä joudutaan tekemään ratkaisuja epävarman tiedon varassa.
- *Episodinen ja ei-episodinen ympäristö.* Episodisessa ympäristössä agentin toiminta koostuu toistuvista jaksoista, joilla ei ole keskenään mitään yhteyttä. Ei-episodisessa ympäristössä taas tilanteet eivät muodosta jaksoja. Esimerkiksi sähköpostinlajitteluagentti toimii episodisessa maailmassa. Episodisessa maailmassa toimiva agentti on suunnittelijan näkökulmasta yksinkertaisempi suunnitella kuin ei-episodisessa maailmassa toimiva agentti. Koska agentti voi päättää toiminnosta perustuen meneillään olevaan episodiin, sen ei tarvitse tietää mitään menneistä tai tulevista episodeista.
- *Staattinen ja dynaaminen ympäristö.* Staattisessa ympäristössä toimiva agentti voi olla varma siitä, että ympäristön tila muuttuu vain agentin toiminnan tuloksena. Dynaamisessa ympäristössä tila voi muuttua ilman, että agentin toiminnot vaikuttavat siihen. Fyysinen maailma on erittäin dynaaminen ympäristö.
- *Diskreetti ja jatkuva ympäristö.* Diskreetti ympäristö sisältää rajallisen määrän mahdollisia toimintoja ja havaintoja. Jatkuvassa ympäristössä erilaisia mahdollisia toimintoja tai havaintoja on rajoittamaton määrä. Shakkipeli on esimerkki diskreetistä ympäristöstä.

Erilaiset ympäristöt vaativat jokseenkin erilaista ongelmanratkaisukykyä. Kaikkein vaikeinta lienee toteuttaa agentti ympäristöön, joka on suljettu, ei-episodinen, dynaaminen ja jatkuva. Jos ympäristö on riittävän monimutkainen, niin se voi käytännössä olla epädeterministinen, vaikka se teoriassa olisikin deterministinen [Russel and Norvig, 1995].

2.1.2 Wooldridge: Agentit ova joustavia

Kuvassa 2.1 esitetyksi agentiksi voi kutsua melkein mitä tahansa jatkuvasti toimivaa tietokoneohjelmaa. Esimerkiksi Unix-käyttöjärjestelmän taustaprosessit voi nähdä agenttina, mutta kovin älykkääksi niitä ei kuitenkaan

voi kutsua. Wooldridgen [1999] mukaan älykäs agentti kykenee *joustavaan* autonomiseen toimintaan täyttääkseen suunnitellut tavoitteet, missä joustavuus koostuu kolmesta ominaisuudesta:

- *reaktiivisuus*: Agentti havainnoi ympäristöään ja reagoi siinä tapahtuviin muutoksiin niin nopeasti kuin mahdollista.
- *proaktiivisuus*: Agentti käyttäytyy tavoitteellisesti ja ottaa joskus aloitteen itselleen saavuttaakseen suunnitellut päämäärät.
- *sosiaalisuus*: Agentti kykenee keskusteluihin toisten agenttien kanssa.

Tämä on niinsanottu agentin heikko käsite [Wooldridge and Jennings, 1994]. Agentin vahvemmassa käsitteessä termillä agentti tarkoitetaan sellaista järjestelmää, joka on toteutettu käyttäen käsitteitä, jotka yleensä yhdistetään ihmisiin. Tällaisia käsitteitä ovat mm. uskomus, tietämys, aikomus ja pakko.

Tasapainon löytäminen proaktiivisen ja reaktiivisen toiminnan kesken on vaikeaa: halutaan, että agentilla on tavoitteita, joita se yrittää saavuttaa, mutta se ei saa tavoitella niitä sokeasti [Wooldridge, 1999]. Jos maailmaa hahmotetaan agenttipohjaisesti, käy nopeasti ilmi, että suurin osa ongelmista vaatii monta agenttia [Jennings, 2000]. Jotta monesta agentista olisi oikeasti hyötyä, niiden on kyettävä keskustelemaan keskenään. Keskustelulla ei tässä yhteydessä tarkoiteta pelkästään tietokoneiden välistä tiedonsiirtoa, jota tapahtuu joka päivä esimerkiksi pankkisovelluksissa. Agenttien kommunikaatio tapahtuu korkeammalla tasolla. Ne neuvottelevat ja tekevät yhteistyötä saavuttaakseen tavoitteensa.

Jotta nähtäisiin, mitä edellä mainituilla ominaisuuksilla tarkoitetaan, kuvitellaan tilanne, jossa öljytankkeri on juuri lähtenyt Primorskin öljyterminaalista Viipurinlahden suulta. Tankkeri on varustettu uusinta teknologiaa edustavalla automaattiperämiehellä (tästä lähtien APM), jonka tavoitteena on ohjata laiva määräsatamaansa Gibraltariin. APM:n tulee saavuttaa annetut reitin etapit tiettyyn aikaan mennessä. Hyvän matkaa kuljettua tulee vastaan navakan tuulen kasaamaa ahtojäätä. On selvää, ettei ole kovin järkevää puskea reittinopeudella jäätikköön, vaan on reagoitava tilanteeseen. Vähintäänkin on hidastettava vauhtia tai jopa pysähdyttävä ja kutsuttava jäänmurtaja paikalle, jotta säästyttäisiin mahdolliselta vakavalta ympäristökatastrofilta. Tilanteesta viisastuneena APM päättää tilata myös uuden jääennusteen saadakseen selville mahdolliset uuden ahtojääpaikat. Näin APM siis joutuu arvioimaan tavoitteensa uudestaan ja valitsemaan jonkin uuden tavoitteen ja mahdollisesti neuvottelemaan jäänmurtaja-agentin kanssa

avun saannista. Uuden jääennusteen valossa se voi suunnitella loppureittinsä Suomenlahdelta pois siten, että se välttää pahimmat jäätiköt.

2.1.3 Muita määritelmiä

Edellä mainitut ominaisuudet eivät kaikkien tutkijoiden mielestä riitä. Esimerkiksi Green *et al.* [1997] esittävät, että agentti on tietojenkäsittelyllinen kokonaisuus, joka

- toimii autonomisesti toisten puolesta,
- osoittaa toiminnoissaan jonkin verran proaktiivisuutta ja/tai reaktiivisuutta, ja
- omaa joitakin tärkeitä ominaisuuksia, jotka liittyvät oppimiseen, yhteistyöhon ja liikkuvuuteen.

Franklin ja Graesser [1996] listaavat erilaisia agentin määritelmiä ja johtavat niistä oman määritelmänsä:

> ”Autonominen agentti on järjestelmä, on sijoitettu johonkin ympäristöön, havainnoi ja suorittaa toimintoja siinä, ajallisesti pitkäänkin, oman agendansa mukaan vaikuttaakseen siihen, mitä se havainnoi tulevaisuudessa. ”

Living Systems on agenttiteknologiaa käyttävä yritys, jossa agenttien nähdään olevan

> ”...ohjelmisto-objekteja, jotka toimivat proaktiivisesti ihmisten puolesta tavoitellen niille delegoituja tavoitteita”[LivingSystems, 2002].

Tämän työn tarkoitus ei ole tuottaa uutta agenttimääritelmää. Tästä eteenpäin agentilla tarkoitetaan kohtien 2.1.1 ja 2.1.2 yhdistelmää, eli ohjelmistokokonaisuutta, joka havainnoi ja toimii autonomisesti ympäristössä ja kykenee joustavaan toimintaan. Nämä kaksi määritelmää kuvaavat agentin ominaisuuksia hyvin. Jos haluamme älykkäitä ohjelmistokomponentteja, jotka tekevät asioita meidän puolestamme, tulee niiden olla autonomisia. Jotta komponentit pärjäisivät autonomisesti, niiden on osattava reagoida ympäristön muutoksiin ja ennakoitava tilanteita. Jos ne eivät pärjää itse, niiden on kyettävä pyytämään apua toisilta agenteilta.

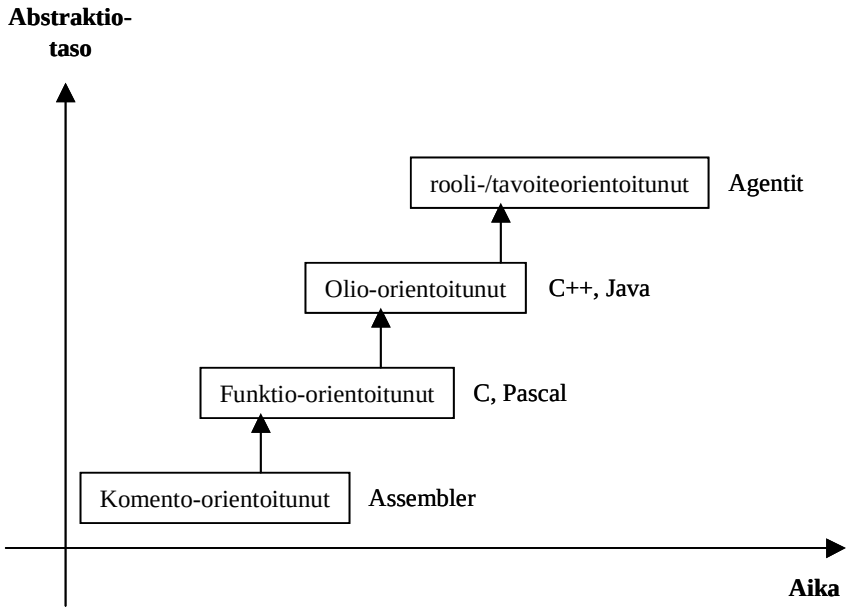
2.2 Agentin autonomisuus

Autonomisuus on olennainen osa agentin olemusta. Sillä tarkoitetaan agenttien yhteydessä jonkinasteista riippumattomuutta ihmisen ohjailusta. Russelin ja Norvigin [1995] mukaan agentti on autonominen, jos sen käyttäytymisen määräävät sen omat kokemukset. He toteavat kuitenkin, ettei tämä tarkoita sitä, että agentin tulisi ensin toimia satunnaisesti, jotta se saisi kokemuksia ja havaintoja ympäristöstään. Agentille siis pitää antaa jonkin verran alustavaa tietämystä, mutta agentti, joka toimii sisäänrakennetuilla oletuksilla, toimii hyvin vain, kun nuo oletukset pitävät paikkansa [Russel and Norvig, 1995]. Siltä puuttuu joustavuutta. Periaatteessa täysin autonomisen agentin tulisi kyetä toimimaan onnistuneesti monissa erilaisissa ympäristöissä, jos sille annetaan aikaa mukautua [Russel and Norvig, 1995].

Odellin [1999] mukaan autonomisuus on parasta ilmaista autonomisuuden asteena, eikä niin, että sitä joko on tai ei ole. Agentit nauttivat siis *jonkinasteisesta* autonomiasta. Dynaaminen autonomia tarkoittaa sitä, että agentti päättää itse, mitä se tekee ympäristössään. Se reagoi joihinkin asioihin ja toimii ennakoivasti joissakin tapauksissa. Epädeterministinen autonomia tarkoittaa sitä, että agentti toimii ympäristöstä katsottuna epädeterministisesti, eli ei voida aina etukäteen tietää, miten agentti toimii kullakin hetkellä. Toteutettavan järjestelmän toiminnan tulos on tietenkin oltava tiedossa, mutta se, miten ja mitkä agentit sen saavat aikaan, ei aina ole suunnitteluhetkellä tiedossa.

2.3 Agenttiohjelmoinnin historiaa

Agenttiohjelmointi ei ole ilmaantunut tyhjästä, vaan se on saanut vaikutteita monista tekniikoista ja tutkimusalueista. Näistä tärkeimmät ovat tekoäly ja olio-ohjelmointi. Myös asiantuntijajärjestelmissä on paljon samaa kuin ohjelmistoagenteissa.



Kuva 2.2 Ohjelmoinnin kehitys [LivingSystems, 2002].

Kuvassa 2.2 näkyy eräs käsitys ohjelmoinnin kehtyksestä. Kehitys on kulkenut komento-orientoituneesta Assembler-kielellä suoritetusta ohjelmoinnista agenttiohjelmointiin. Jokainen kehitystaso on kohottanut abstraktiotasoa edelliseen nähden. Kuvassa agenttiohjelmointi on korkeimmalla abstraktiotasolla. Se on rooli- ja tavoiteorientoitunutta. Agenttiohjelmoinnissa järjestelmiä kuvataan sellaisilla käsitteillä, kuin uskomus, toive, aikomus, tahto ja motiivi. Järjestelmän kuvailemista tällaisten käsitteiden kautta kutsutaan intentionaaliseksi asennoitumiseksi (intentional stance). Se on hyvä abstraktiotyökalu silloin, kun puhutaan erittäin monimutkaisista järjestelmistä, joiden toimintaa ei kokonaisuudessaan ymmärretä [Wooldridge, 1997].

2.3.1 Agentit ja tekoäly

Tekoäly on epäilemättä suurin agenttitutkimukseen vaikuttanut alue. Tekoälyn osa-alueista varsinkin suunnittelu on erittäin läheisesti yhteydessä agentteihin. Suunnittelussa pyritään etsimään ratkaisua sille, mitä pitää tehdä: mikä toiminto pitää suorittaa [Jennings et al., 1998]. Se juontaa juurensa Newellin ja Simonin [Newell and Simon, 1963] GPS-järjestelmään (General Problem Solver). Yleensä se kuitenkin yhdistetään STRIPS-suunnittelijaan [Fikes and Nilsson, 1971]. Tyypillisesti STRIPS-järjestelmä sisältää ainakin seuraavat osat [Jennings et al., 1998]:

1. symbolisen mallin agentin ympäristöstä,
2. symbolisen kuvauksen agentin käytettävissä olevista toiminnoista, sekä
3. suunnittelualgoritmin.

Suunnittelualgoritmi saa syötteenään ympäristön kuvauksen, joukon toimintokuvauksia sekä tavoitetilan kuvauksen. Lopputuloksena se tuottaa suunnitelman.

Suunnittelujärjestelmät siis luovat suunnitelman, joka koostuu peräkkäin suoritettavista toiminnoista. Kun suunnitelman toiminnot suoritettaan, saavutetaan haettu tavoitetila. Toimintaperiaatteena on, että suunnittelujärjestelmä toimii silmukassa, jossa valitaan ensin tavoite, sitten generoidaan suunnitelma, suoritetaan suunnitelma, valitaan uusi tavoite jne. Tällainen *havainnoi-suunnittele-toimi* -tyyppinen ongelmanratkaisu perustuu kokonaisuudessaan symboliseen esitykseen ja päättelyyn [Jennings et al., 1998].

Suunnittelijat (planner), kuten STRIPS, tuottavat periaatteessa aina oikean ja todistettavissa olevan suunnitelman, jos sellainen on olemassa. Mutta ne eivät välttämättä saa tuotettua sellaista vaaditussa ajassa, jolloin tuotetusta suunnitelmasta ei ole hyötyä. Tämä havainto johti monet tutkijat epäilemään koko symbolisen tekoälyn oikeutusta [Jennings, 1998]. Tunnetuin tämän suunnan edustaja on R. Brooks, joka kehitti ns. reaktiivisen lähestymistavan agenttien toteutuksen. Hänen mielestään [1991] älykkyys on agentin ja ympäristön vuorovaikutuksen tulos. Lisäksi älykkyys ilmaantuu erilaisten yksinkertaisten käyttäytymisten vuorovaikutuksesta [Brooks, 1991]. Osana tutkimustaan hän kehitti reaktiivisen agenttiarkkitehtuurin [Brooks, 1985; 1991], josta esitän lyhyen kuvauksen luvussa neljä.

Myöhemmin havaittiin, että reaktiivisetkaan järjestelmät eivät sovi kaikkiin tilanteisiin. Tarvitaan sekoitus reaktiivisuutta ja symbolista päättelyä. Tämä johti erilaisten kerrosarkkitehtuurien kehittelyyn, joissa alimmat kerrokset huolehtivat reaktiivisimmista ominaisuuksista ja ylemmät suunnittelevat toimintaa symbolisella päättelyllä. Tästä lähestymistavasta on kaksi esimerkkiä luvussa neljä.

Käytännön järkeily (practical reasoning) on vaikuttanut myös paljon agenttiarkkitehtuurien kehitykseen. Käytännön järkeilyllä tarkoitetaan pragmaattista järkeilyä, jota käytetään päätettäessä mitä tehdään. Tunnetuin käytännön järkeilyyn perustuva teoria on BDI-teoria. Siinä agentin päätöksenteko perustuu uskomuksien (belief), toiveiden (desire) ja intentioiden (intention) vuorovaikutukseen. Luvussa neljä esittelen kaksi BDI-teoreettista agenttiarkkitehtuuria.

2.3.2 *Agentit ja oliot*

Tekoälyn lisäksi myös olio-ohjelmointi on vaikuttanut agentteihin. Olioilla ja agenteilla on paljon yhteistä. Olio on rajattu, identiteetin omaava kokonaisuus, jolla on oma identiteetti, tila ja käyttäytyminen [OMG, 2003]. Olion atribuutit määrittelevät sen tilan ja metodit sen käyttäytymisen. Myös agentin voidaan ajatella olevan olio, koska silläkin on oma identiteetti, tila ja käyttäytyminen [OMG, 2000]. Molemmat voivat olla minkä kokoisia tahansa, mutta yleensä pyritään suunnittelemaan pieniä agentteja ja olioita [OMG, 2000].

Olioilla ja agenteilla on kuitenkin perustavaa laatua olevia eroja [OMG, 2000; Wooldridge, 1999]:

1. Agentit ovat autonomisempia, kuin oliot. Agentit päättävät itse toimintojensa suorittamisesta. Oliot suorittavat toiminnon, kun joku muu kutsuu niiden metodeja. Poikkeuksena tähän on aktiivinen olio, joka toimii omassa säikeessään ja on näin jossakin määrin autonominen.
2. Agentit kykenevät joustavaan (reaktiiviseen, proaktiiviseen, sosiaaliseen) toimintaan, mihin olioparadigmassa ei oteta kantaa.
3. Agentit kommunikoivat keskenään lähettämällä toisilleen kommunikaatiokielellä kuvattuja viestejä, oliot kommunikoivat kutsumalla toistensa metodeja.
4. Olioperustaisissa ohjelmistoissa kaikki oliot ovat jonkin luokan ilmentymiä. Agentit voivat olla tyypitettyjä, mutta kaikki agentit eivät ole. Sama pätee perimiseen.
5. Oliopohjaisissa järjestelmissä olioilla ajatellaan olevan jonkinlainen yhteinen tavoite, agenteilla näin ei välttämättä ole.

Odellin [1999] mukaan olio kapseloi identiteettinsä (kuka), tilansa (mitä) ja toimintansa (kuinka). Aktiivinen olio kapseloi oman kontrollisäikeensä (milloin). Agentti kapseloi edellämainittujen ominaisuuksien lisäksi sen, *miksi* se tekee jotain.

Agentin ja olion erot ovat joskus aika pieniä, koska agentit toteutetaan usein oliokielellä, kuten esimerkiksi Javalla ja C++:lla. On kuitenkin tärkeätä tehdä ero agentin ja olion välille, koska ne toimivat eri abstraktiotasolla. Lisäksi oliot sopivat paremmin joidenkin ja agentit taas joidenkin toisten asioiden tekemiseen.

2.3.3 Agentit ja asiantuntijajärjestelmät

Asiantuntijajärjestelmä on tietokoneohjelma, joka sisältää jonkin rajatun alan erikoistietämystä ja tekee päätelmiä tästä tietämyksestä. Tarkoitus on matkia ihmisasiantuntijan kykyä esittää kysymyksiä, selittää, miksi ne kysyttiin, tehdä johtopäätöksiä ja puolustaa tehtyjä johtopäätöksiä [Gero and Stanton, 1987].

Asiantuntijajärjestelmän kaksi pääosaa ovat tietämyskanta ja päättelykone. Tietämyskanta sisältää sovellusalan tietämyksen ja päättelykone tekee johtopäätöksiä tietämyskannan tietämyksestä.

Agentilla voi hyvinkin olla asiantuntijajärjestelmän kaltaisia ominaisuuksia, mutta asiantuntijajärjestelmä ei ole agentti. Tärkein ero asiantuntijajärjestelmän ja agenttien välillä on, että agentit toimivat jossakin ympäristössä, kun taas asiantuntijajärjestelmät eivät toimi vuorovaikutuksessa ympäristön kanssa [Wooldridge, 1999]. Agenttitutkijoiden tavoitteena on rakentaa älykkäitä *toimijoita*, ei vain älykkäitä ajattelijoita [Pollack, 1992]. Asiantuntijajärjestelmät eivät saa sensoreilta syötteitä, vaan syötteet tulevat ihmisen välityksellä. Niiden ei myöskään tarvitse kommunikoida toisten asiantuntijajärjestelmien kanssa, mikä taas on agenteille tyypillistä [Wooldridge, 1999].

2.4 Yhteenveto

Agentin määrittely on osoittautunut vaikeaksi suorittaa. Kahden yleisesti käytetyn määritelmän yhdistelmänä totean, että agentti toimii ympäristössä sekä kykenee joustavaan ja autonomiseen toimintaan.

Agentin erilaiset ympäristöt voidaan jaotella ryhmiin ympäristöjen ominaisuuksien mukaan. Ympäristö vaikuttaa agentin päätöksentekomekanismin monimutkaisuuteen. Kaikkein vaikeinta on toteuttaa agentti ympäristöön, joka on suljettu, ei-episodinen, dynaaminen ja jatkuva. Agentti nauttii jonkinasteista autonomiaa toimimassaan ympäristössä.

Agenttiohjelmointi on saanut vaikutteita pääasiassa tekoälyn ja olio-ohjelmoinnin menetelmistä. Voidaankin todeta, että agentti on usein autonominen, älykäs olio.

3 Rationaalinen toiminta

Rationaalinen toiminta on yksi agentin keskisimmistä ominaisuuksista. Russel ja Norvig [1995] sijoittavat agentin laatimassaan tekoälyn osa-alueiden nelikentässä rationaalisesti toimiviin järjestelmiin (kuva 3.1).

Ihmisten lailla ajattelevat järjestelmät	Rationaalisesti ajattelevat järjestelmät
Ihmisten lailla toimivat järjestelmät	Rationaalisesti toimivat järjestelmät

Kuva 3.1. Tekoälyn osa-alueet [Russel and Norvig, 1995].

Myös Bratman et al. [1988], Wooldridge [2000] sekä Hoek ja Wooldridge [2003] näkevät rationaalisen toiminnan agentin oleelliseksi ominaisuudeksi. Wooldridgen mukaan [1996] on tärkeää kuitenkin erottaa tekoälytutkimuksen lopullisena tavoitteena oleva älykkyys laajemmassa mielessä siitä, mitä vaaditaan agenteilta. Ainoa agenteille asetettava älykkyyden vaatimus on, että ne tekevät riittävän nopeasti hyväksyttävissä olevia päätöksiä suoritettavista toiminnoista.

Rationaalisella toiminnalla tarkoitetaan sitä, että agentti valitsee suoritettavaksi uskomustensa valossa sille itselleen parhaita toimintoja [Wooldridge, 2000]. Russelin ja Norvigin [1995] mukaan se, mikä on paras toiminto milläkin ajanhetkellä, riippuu seuraavista seikoista:

1. suoritusmittarista, jolla onnistumisen astetta mitataan,
2. agentin havaintohistoriasta,
3. siitä, mitä tiedetään toimintaympäristöstä, ja
4. siitä, mitä toimintoja agentti osaa suorittaa.

Agentin ei tarvitse yleensä tietää toimenpiteidensä lopputulosta toimiakseen rationaalisesti. Agentti voi epäonnistua, vaikka se toimisi rationaalisesti. Epäonnistuminen on oleellinen osa agenttien toimintaa. Esimerkiksi WWW-sivun muutoksia seuraava agentti ei voi varmasti tietää, onko sivua enää olemassa, kun agentti yrittää hakea sen. Jos agentin uskomus

ennen toimintoa oli, että sivu on saatavissa, on toiminta rationaalista huolimatta lopputuloksesta.

Rationaalinen toiminta on ollut tutkimuksen kohteena myös monilla muilla tieteenaloilla, kuten esimerkiksi taloustieteissä, filosofiassa ja kognitiivisissa tieteissä [Hoek and Wooldridge, 2003].

3.1 Joitakin rationaalisen toiminnan lähtökohtia

Rationaalista toimintaa voidaan saada aikaan monella eri tavalla. Päätöksentekoteoriassa rationaalisuus tulee hyödyn maksimoinnista, samoin peliteoriassa. Käytännön järkeilyssä pyritään löytämään oikeat keinot tietyn lopputuloksen saavuttamiseksi.

3.1.1 Päätöksentekoteoria

Päätöksentekoteoriassa (decision theory) rationaalinen agentti nähdään sellaisena, joka maksimoi sille koituvan odotettavissa olevan hyödyn [Russel and Norvig, 1995]. Päätöksentekoteoreettisella agentilla on hyötyfunktio, jolla se laskee eri tiloista itselleen koituvan hyödyn numeerisena arvona. Oletuksena on, että agentin suorittama toiminto voi tuottaa tuloksenaan monta vaihtoehtoista tilaa. Agentti laskee näiden vaihtoehtoisten tulosten todennäköisyyden ennen toiminnon suorittamista. Kun tulosten todennäköisyydet lasketaan yhteen niiden tuottamien tilojen hyötyjen kanssa, saadaan toiminnon odotettavissa oleva hyöty. Maksimaalisen odotettavissa olevan hyödyn periaatteen mukaan rationaalisen agentin tulee valita se toiminto, jonka odotettavissa oleva hyöty on suurin [Russel and Norvig, 1995].

Vaikka maksimaalisen odotettavissa olevan hyödyn periaate määrittelee oikean toiminnon missä tahansa tilanteessa, käytännössä sen vaatimat todennäköisyyksien ja hyödyn laskennat voivat olla erittäin raskaita. Maksimaalisen hyödyn laskenta tarkoittaa nimittäin sitä, että joudutaan etsimään kaikkien toimintojen kaikki lopputulokset [Wooldridge, 2002]. Lisäksi eri lopputulosten todennäköisyysjakaumien määrittely on vaikeaa ja voi olla laskennallisesti kallista [Russel and Norvig, 1995].

3.1.2 Peliteoria

Peliteoriassa tutkitaan päätöksentekoa sellaisessa tilanteessa, jossa pelaajat joutuvat tekemään päätöksiä, jotka potentiaalisesti vaikuttavat toisten pelaajien intresseihin [Turocy and Stengel, 2002]. Sen matemaattisesta perustasta johtuen sitä käytetään automaattisten päätöstentekoprosessien mallintamisen interaktiivisissa ympäristöissä, kuten esimerkiksi neuvottelussa ja huutokaupassa. Peliteorian keskeinen käsite on peli, joka on formaali malli

interaktiivisesta tilanteesta. Pelin lisäksi määritellään pelaajat sekä heidän informaationsa, preferenssinsä, käytettävissä olevat strategiat ja kuinka nämä kaikki vaikuttavat lopputulokseen [Turocy and Stengel, 2002].

Yhteistyöpeli kuvaa korkealla tasolla, mitä palkintoja mikäkin potentiaalinen joukko tai koalitio voi saavuttaa, jos sen jäsenet tekevät yhteistyötä. Pelissä pyritään löytämään mahdollisimman hyvä koalitio. Kilpailevassa pelissä tutkitaan toisten pelaajien itsekkäistä syistä tekemiä strategisia valintoja. Kilpaileva peli voi muuttua yhteistyöksi, jos se on pelaajien mielestä heidän henkilökohtaisten etujensa mukaista. Rationaalinen pelaaja valitsee toiminnon, joka maksimoi sille koituvan hyödyn. Valinta tehdään sen perusteella, mitä pelaaja odottaa toisten pelaajien tekevän.

Peliteorian vahvuus on matemaattinen metodologia, jolla voidaan tutkia ja analysoida strategisia valintoja. Kun tilanne mallinnetaan formaaliksi peliksi, joutuu päätöksentekijä päätöksentekijä luettelemaan eksplisiittisesti pelaajat ja heidän strategiset vaihtoehtonsa sekä ottamaan huomioon heidän preferenssinsä ja reaktionsa [Turocy and Stengel, 2002]. Tämä tuo päätöksentekijälle laajemman ja selvemmän näkökulman tilanteeseen.

3.1.3 Käytännön järkeily

Käytännön järkeily on toimintaan johtavaa suunnittelemista. Siinä on kaksi vaihetta. Ensimmäisessä vaiheessa harkitaan mitä halutaan saavuttaa (deliberation). Toisessa vaiheessa etsitään keinoja, sen saavuttamiseksi (means-ends reasoning). Esimerkiksi jos haluan matkustaa Helsinkiin, minulla on kaksi vaihtoehtoa: matkustaa autolla ja matkustaa junalla. Harkinnan lopputuloksena päätän matkustaa junalla (esimerkiksi, koska en ole vaihtanut talvirenkaita autoon). Kun olen päättänyt matkustaa junalla, minun tulee etsiä keinot junalla Helsinkiin pääsemiseksi. Tämän keinojen etsimisen lopputuloksena minulla on suunnitelma, joka vie minut ensin rautatieasemalle ja myöhemmin junalla Helsinkiin. Kun minulla on suunnitelma, toteutan sen. Suunnitelman suorituksen jälkeen minun tulisi olla Helsingissä.

Käytännön järkeily on ollut tutkimuskohteena filosofien keskuudessa pitkään. Jo Aristoteleen mukaan ihmiset olettavat päämäärän ja etsivät keinoja sen saavuttamiseksi [Russel and Norvig, 1995]. Kun keino on löytynyt, etsitään keino sen löytämiseksi ja niin edelleen. Newell ja Simon toteuttivat Aristoteleen lähestymistavan General Problem Solver -järjestelmällään. Tunnetumpi samaan menetelmään perustuva järjestelmä on aiemmin mainittu STRIPS-suunnittelija [Jennings, 1998].

Bratmanin [1990] mukaan käytännön järkeilyssä punnitaan kilpailevia vaihtoehtoja puoltavia ja vastustavia näkökohtia. Näkökohdat tulevat siitä, mitä agentti haluaa/arvostaa ja siitä, mistä se välittää.

On tärkeää erottaa käytännön järkeily teoreettisesta järkeilystä. Jälkimmäinen kohdistuu tietoon ja se pyrkii lähinnä saamaan selville jonkin proposition totuusarvon [Anscombe, 1978]. Teoreettista järkeilyä on esimerkiksi vanha päätelmä: jos uskon, että jokainen ihminen on kuolevainen ja että Sokrates on ihminen, voin päätellä, että Sokrates on kuolevainen. Päättelyn tulos ei siis johda toimintaan.

3.2 BDI-malli

Kirjainyhdistelmä BDI tulee sanoista beliefs (uskomukset), desires (toiveet) ja intentions (aikomukset, intentiot). BDI-malli perustuu Michael Bratmanin kehittämään teoriaan rationaalisen agentin (jollainen ihminenkin on) toiminnasta. Tämä teoria perustuu käytännölliseen järkeilyyn, jota käytämme joka päivä päättäessämme tekemisistämme ja suunnitelmistamme.

Agentin uskomukset vastaavat informaatiota, jota sillä on ympäristöstään. Agentin toiveet kuvaavat niitä maailman tiloja, jotka agentti teoriassa haluaisi saada aikaan. Ihmisillä toiveet voivat olla keskenään ristiriitaisia. Keinotekoisilla agenteilla toiveiden oletetaan yleensä olevan keskenään johdonmukaisia. Agentin intentiot kuvaavat niitä toiveita, jotka agentti on sitoutunut saavuttamaan.

3.2.1 Intentiot ja käytännön järkeily

Intentiota käytetään kuvaamaan sekä mielentilaa, että toimintaa [Bratman, 1990]. Voidaan sanoa, että kirjoitin tämän kappaleen intentionaalisesti ja että intentioni on ollut selventää käytettyä käsitteistöä. Voidaan myös sanoa, että kirjoitan tätä kappaletta, jotta intentioni valmistua piakkoin toteutuisi. Ensimmäisessä tapauksessa intentio kuvaa tiettyä mielentilaa, jälkimmäisessä taas toimintaa.

Intentiot voivat kohdistua joko tulevaisuuteen tai nykyhetkeen [Bratman, 1990]. Tulevaisuuteen kohdistuvat intentiot ohjaavat agentin suunnittelua ja uusien intentioden valintaa, nykyhetkeen kohdistuvat intentiot puolestaan toimivat kausaalisesti tuottaen käyttäytymistä. Tulevaisuuteen kohdistuva intentio voi olla esimerkiksi uimaan meneminen huomenna ja nykyhetkeen kohdistuva käden liikuttaminen. Bratmanin mallissa intentioilla käsitetään pääasiassa tulevaisuuteen kohdistuvia aikomuksia.

Bratmanin [1990] mukaan tulevaisuuteen suuntautuvat intentiot ovat tärkeitä, koska toiminnan harkinta vie aikaa ja muita resursseja. Tämä tarkoittaa, että harkinnalle on olemassa jokin takaraja, milloin viimeistään pitää päättää, mitä tehdään. Tällöin tyytyminen tulevaisuuteen suuntautuvaan intentioon vähentää tarvittavan harkinnan määrää tulevaisuudessa. Toinen syy on, että tulevaisuuteen suuntautuvat intentiot auttavat koordinoimaan toimintaa.

Intentioiden ilmeisin ominaisuus on, että ne johtavat toimintaan. Kun aika kuluu, tulevaisuuteen suuntautuvat intentiot muuttuvat nykyhetkeen suuntautuviksi intentioiksi, jolloin ne tuottavat toimintaa. Tietenkin olosuhteet voivat muuttua, joten intentiot eivät aina johda toimintaan. Intentioilla on monta muutakin ominaisuutta [Bratman, 1990; Cohen and Levesque, 1990]:

1. Intentiot tuovat normaalisti ongelmia käytännön järkeilyyn, koska agentin täytyy löytää tavat, joilla ne saavutetaan.
2. Intentiot rajoittavat uusien intentioiden valintaa. Vaikka toiveet voivat olla ristiriitaisia, agentti ei yleensä valitse intentioita, jotka ovat ristiriidassa nykyisten intentioiden kanssa.
3. Intentiot pysyvät. Jos yritys epäonnistuu, yritetään uudestaan tai etsitään toinen vaihtoehtoinen suunnitelma. Jos intentio osoittautuu mahdottomaksi saavuttaa (esim. toiminnan viimeinen sallittu ajankohta on jo ylitetty eikä toimintoa ole vielä suoritettu), siitä luovutaan.
4. Intentiot vaikuttavat tulevaisuuden uskomuksiin, joten agentit voivat suunnitella tulevaisuuttaan sillä oletuksella, että niiden intentiot saavutetaan.
5. Intentioiden tiedossa olevista sivuvaikutuksista ei tarvitse muodostaa intentioita. Agentti *valitsee* intention lisäksi myös sen sivuvaikutukset [Bratman, 1990], kuten esimerkiksi hampaan paikkauksen aiheuttaman kivun. Se ei kuitenkaan seuraa näiden valittujen sivuvaikutusten tilaa, kuten se tekee intentioiden suhteen. Agentin intentio siis ei ole kokea kipua (paitsi, jos kyseessä on masokistinen agentti, kuten elokuvassa ”Pieni kauhukauppa”), vaan se valitsee paikkauksen lisäksi myös kivun. Jos agentille selviää, että paikkaus ei olekaan kivulias, se ei jätä intentiotaan saada hampaansa paikatuksi.

3.2.2 Suunnitelmat

Suunnitelmat ja intentiot ovat hyvin samanlaisia monessa suhteessa. Esimerkiksi lauseilla ”Suunnittelen tekeväni....” ja ”Aion tehdä....” on sama merkitys. On olemassa kahdenlaisia suunnitelmia [Pollack, 1990]:

1. Suunnitelmia, jotka ovat reseptejä jonkin toiminnon suorittamiseen tai tavoitteen saavuttamiseen.
2. Suunnitelmia, jotka agentti ottaa suoritettavakseen ja jotka myöhemmin ohjaavat sen toimintaa.

Ensimmäisessä tapauksessa agentti tietää, että on olemassa suunnitelma, joka vie junalla Helsinkiin. Toisessa tapauksessa agentti oikeasti suunnittelee matkustavansa junalla Helsinkiin. Rajallisilla resursseilla varustetun agentin suunnitelmat ovat osittaisia. Jos agentti suunnittelee matkustavansa ensi viikolla Helsinkiin, sen ei tarvitse suunnitella koko matkaa heti. Se voi tyytyä tällä hetkellä suunnitelmaan, että se on menossa Helsinkiin ja harkita myöhemmin, miten se pääsee sinne. Suunnitelmat ovat lisäksi hierarkkisia. Päämääriä sisältäviin suunnitelmiin on lisätty päämääriä saavuttavia suunnitelmia [Bratman, 1990].

Kun sitoudutaan etukäteen osittaisiin suunnitelmiin, säästyy aikaa. Lisäksi maailma muuttuu nopeasti, joten kauas tulevaisuuteen suuntautuvat yksityiskohtaiset suunnitelmat ehtivät vanhentua, ennen kuin niitä päästään suorittamaan [Bratman, 1990].

3.2.3 Sitoutuminen

Kun agentti sitoutuu intentioon, se sitoutuu sekä päämäärään, että keinoihin. Edellä on todettu, että intentiot pysyvät ja että se yrittää uudestaan epäonnistumisen jälkeen. BDI-agenttien toteutuksessa on tärkeää määritellä, koska intentiosta voi luopua. Agenteilla on kolmenlaisia sitoutumisstrategioita [Wooldridge, 2000]:

1. Sokeasti sitoutuva agentti ylläpitää intentiota, kunnes se uskoo, että intentio on saavutettu.
2. Hajamielisesti sitoutuva agentti ylläpitää intentiota, kunnes se huomaa, että intentio on joko saavutettu tai että sitä ei ole mahdollista saavuttaa.
3. Avoimin mielin sitoutuva agentti ylläpitää intentiota niin kauan, kuin se uskoo sen olevan mahdollista.

3.3 Yhteenveto

Rationaalinen toiminta on tärkeä päämäärä agenttien kehittämisessä. Rationaalista toimintaa voidaan saada aikaan monilla eri tavoilla. Päätöksentekoteoria ja peliteoria perustuvat hyödyn maksimointiin. Käytännön järkeilyssä pyritään löytämään keinoja päämäärien saavuttamiseksi.

Yksi suosituimmista rationaalisen toiminnan teorioista on BDI-teoria. Siinä sovelletaan uskomuksia, toiveita ja ennen kaikkea intentioita käytännön järkeilyyn. Intentiot tuovat agentille ongelmia, koska sen täytyy löytää keinot niiden saavuttamiseksi.

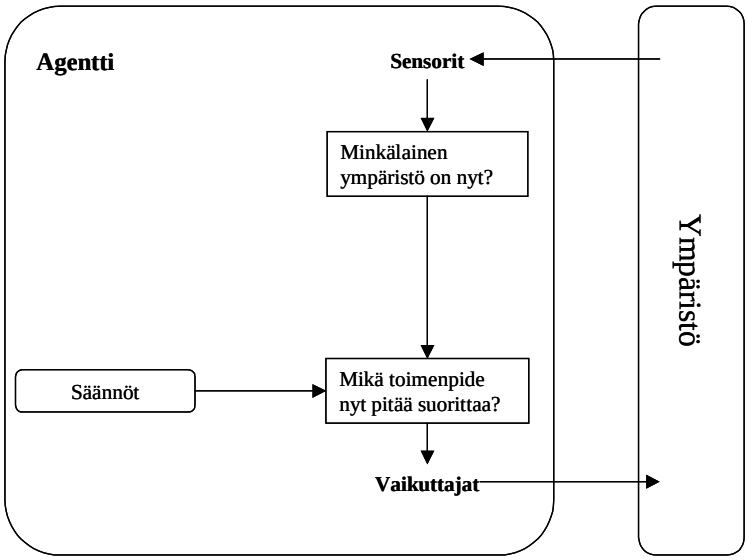
4 Arkkitehtuureja agentin kontrolliin

Agenttiarkkitehtuurilla tarkoitetaan metodologiaa, jolla agentteja rakennetaan. Tarkemmin ilmaistuna tämä tarkoittaa tiettyä tietorakenteiden järjestelyä, algoritmeja ja kontrollivirtoja, joita agentti käyttää päättelyssään. Agenttiarkkitehtuurit voidaankin jaotella löyhästi eri ryhmiin niiden päätöksentekomenetelmän mukaan.

Reaktiivisissa arkkitehtuureissa toimitaan periaatteessa erilaisten sääntöjen varassa. Siinä käytetä minkäänlaista symbolista päättelyä. Kerrosarkkitehtuureissa ja BDI-arkkitehtuureissa agentilla on yksi tai useampi tietämyskanta, joka sisältää tietämystä agentin ympäristöstä ja agentista itsestään. Vaihtoehtoinen ryhmittelytapa olisikin jakaa arkkitehtuurit symbolista päättelyä käyttäviin ja käyttämättömiin, mutta tämän tutkielman lähtökohdista ensin mainittu ryhmittely sopii paremmin.

4.1 Reaktiiviset arkkitehtuurit

Puhtaasti reaktiivisessa arkkitehtuurissa agentti päättää toimistaan ilman tietämystä menneisyydestä. Kuvassa 4.1 näkyy reaktiivisen agentin kaavakuva. Agentti päättää toiminnastaan ”tilanne → toiminto” -tyyppisillä säännöillä agentin tämänhetkisen tilan perusteella. Tällainen päättely ei vaadi symbolista päättelyä.



Kuva 4.1 Puhtaasti reaktiivinen agentti [Russel and Norvig, 1995].

4.1.1 Sisällyttämis-arkkitehtuuri

Brooks kehitti sisällyttämisarkkitehtuurin (subsumption architecture) osana tutkimustaan, joka kritisoi symbolista tekoälyä. Hänen mielestään [1991] todellinen älykkyys syntyy kyvystä liikkua ja suorittaa eloonjäämiseen liittyviä toimintoja dynaamisessa ympäristössä.

Brooksin [1991] mukaan tekoälytutkijat syyllistyivät siihen, että he jakoivat ongelman tekoälyosaan, jonka he ratkaisivat ja ei-tekoälyosaan, jota he eivät ratkaisseet. Jälkimmäinen osa on vastuussa kaikesta havainnoinnista ja motorisista taidoista ja on erittäin tärkeä osa älykästä toimintaa. Brooks [1991] kritisoi lisäksi sitä, että robotteja testattiin ns. leikkimaailmoissa, joita oli yksinkertaistettu huomattavasti. Brooksin [1991] mielestä ei tarvita mitään maailman mallia, vaan on tulee käyttää maailmaa omana mallinaan.

Subsumption-arkkitehtuuri on jaettu aktiviteetteja tuottaviin tilakoneisiin. Ne toimivat yksinkertaisilla ”tilanne → toiminto” -tyyppisillä säännöillä, jotka päättävät agentin tilan perusteella suoritettavan toiminnon. Tilakoneet on järjestetty kerroksiin siten, että alin kerros vastaa primitiivisimmistä toiminnoista, kuten yllättävien esteiden väistämisestä. Ylemmät kerrokset vastaavat monimutkaisemmista toiminnoista, kuten suunnistamisesta. Kun yksi kerros on testattu todellisessa ympäristössä, voidaan lisätä uusi abstraktimpi kerros sen päälle. Alempi kerros ei tiedä ylemmän kerroksen olemassa olosta ja ylempi kerros voi ohjata alemman kerroksen toimintaa esimerkiksi vaimentamalla alemman kerroksen tilakoneiden tuloksia.

Agentilla ei ole keskitettyä kontrollia eikä mitään esitystapaa maailmasta, vaan toiminta perustuu toisiinsa yhdistettyihin tilakoneisiin ja niiden sääntöihin. Älykkyys muodostuu näiden sinänsä yksinkertaisten toimintojen vuorovaikutuksesta.

Sisällyttämisarkkitehtuuriin ja muihin samankaltaisiin liittyy joitakin ongelmia [Jennings et al., 1998]:

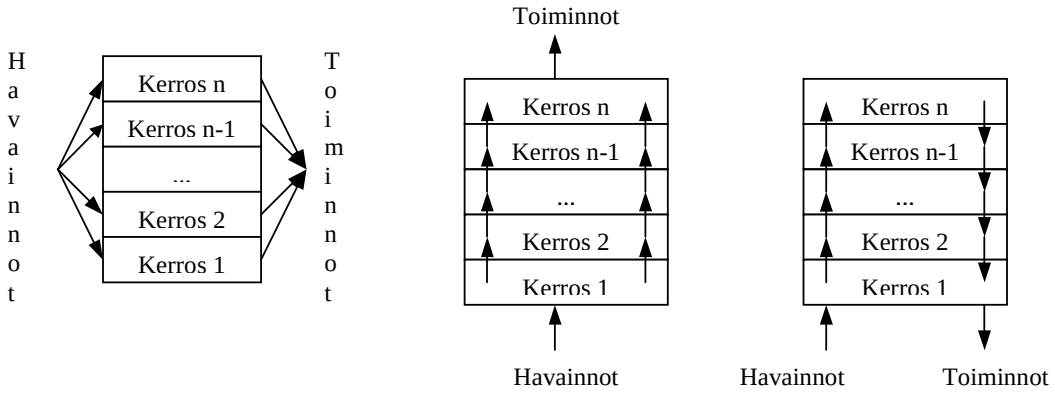
- Jos agentit eivät käytä mallia ympäristöstänsä, niillä täytyy olla riittävästi informaatiota paikallisesti, jotta ne voisivat valita hyväksyttävän toiminnon.
- Kuinka puhtaasti reaktiivinen agentti ottaa huomioon ei-paikallisen informaation, kun se tekee päätöksensä pelkästään paikallisen informaation perusteella?
- Kuinka robotti esimerkiksi osaa suunnistaa toiseen kerrokseen, jos se ei tiedä, mitä toinen kerros tarkoittaa ja mistä sinne pääsee?

- Kuinka puhtaasti reaktiiviset agentit voivat oppia kokemuksistansa ja parantaa suoritustaan ajan mittaan?

Sisällyttämisarkkitehtuurissa yhden tilakoneen toteuttaminen on helpohkoa, samoin alempien tasojen. Kun tasoja on monta, alkavat niiden väliset suhteet olla melkoisen monimutkaisia. Uuden tason lisääminen ja varsinkin sen testaaminen oikeassa maailmassa on tällöin erittäin vaikeaa.

4.2 Kerrosarkkitehtuurit

Kerrosarkkitehtuurissa agentin eri käyttäytymismallit, kuten esimerkiksi havaintojen ja toimintojen suorittaminen, reagointi ympäristön muutoksiin sekä yhteistyö, on jaettu kerroksiin [Müller et al, 1994]. Tyypillisesti kerroksia on ainakin kaksi: toinen huolehtii reaktiivisuudesta ja toinen proaktiivisuudesta, mutta niitä voi olla periaatteessa useitakin [Wooldridge, 1999]. Yleisesti ottaen kerrosarkkitehtuurit voidaan jakaa kahteen pääluokkaan: horisontaalisiin ja vertikaalisiin kerrosarkkitehtuureihin [Müller et al, 1994].



a) Horisontaalinen arkkitehtuuri

b) ja c) Erilaisia vertikaalisia arkkitehtuureja

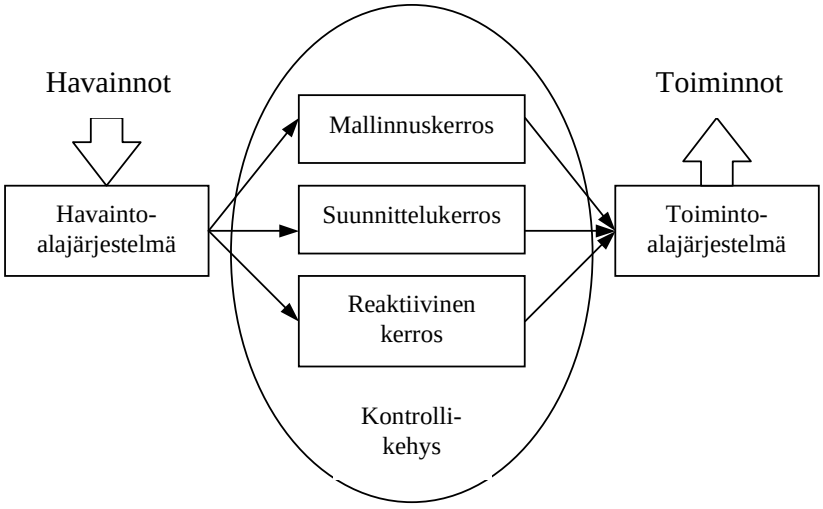
Kuva 4.2. Horisontaalinen ja kaksi vertikaalista kerrosarkkitehtuuria.

Horisontaalisessa kerrosarkkitehtuureissa (kuva 4.2 (a)) ohjelmiston kerrokset ovat suoraan yhteydessä syötteitä ja toimintoja tuottaviin komponentteihin. Jokainen kerros voidaan myös ajatella tavallaan agentiksi, joka tuottaa ehdotuksia siitä, mikä toiminto pitäisi suorittaa [Wooldridge, 1999]. Vertikaalisesssa kerrosarkkitehtuurissa havainnot ja toiminnot suoritetaan korkeintaan yhdessä kerroksessa. Kuvan 4.2 (b) yksisuuntaisessa

kerrosarkkitehtuurissa kontrolli virtaa alhaalta ylös ja ylin taso tuottaa suoritettavia toimintoja. Kuvan 4.2 (c) arkkitehtuurissa tieto virtaa alhaalta ylös ja kontrolli alas. Kaksisuuntainen kerrosarkkitehtuuri muistuttaa toimivaa organisaatiota, kuten armeijaa, jossa tieto virtaa ylös ja käskyt tulevat alas.Horisontaalisen arkkitehtuurin etuna on sen käsitteellinen yksinkertaisuus: jos halutaan agentin käyttäytyvän n:llä eri tavalla, niin toteutetaan n eri kerrosta [Wooldridge, 1999]. Haittapuolena on, että kerrokset tavallaan kilpailevat keskenään. Tällöin tarvitaan keskitetty kontrolliyksikkö, joka ratkaisee eri kerrosten väliset ristiriidat. Tällainen kontrolliyksikkö muodostuu käytännössä pullonkaulaksi horisontaalisessa arkkitehtuurissa [Müller et al, 1994].

4.2.1 TouringMachines

Kuvassa 4.3 on esitetty kaavakuva TouringMachines –arkkitehtuurista. TouringMachines koostuu kolmesta kontrollikerroksesta: reaktiivisesta kerroksesta, suunnittelukerroksesta ja mallinnuskerroksesta. Kukin kerros toimii itsenäisenä yksikkönään ja on liitoksissa havaintojärjestelmään sekä toimintoja suorittavaan alajärjestelmään. Kerrokset mallintavat maailmaa kukin omalla abstraktiotasollaan ja tuottavat ehdotuksia suoritettavista toimenpiteistä kontrollikehykselle.



Kuva 4.3. TouringMachines -arkkitehtuuri [Ferguson, 1992].

Reaktiivinen kerros tarjoaa agentille nopeat toimintamallit äkillisiin muutoksiin ympäristössä, joihin ylemmillä tasoilla ei ole laskennallisia resursseja suunnitella vastinetta. Tyypillinen esimerkki reaktiivisessa

kerroksessa hoidettavasta tilanteesta on toisen agentin tai esteen ilmestyminen näköpiiriin.

Suunnittelukerroksessa suunnitellaan, kuinka asetetut tavoitteet saavutetaan. Tätä tarkoitusta varten sillä on joukko alustavia hierarkkisesti järjestettyjä suunnitelmia, joita voidaan soveltaa eri tilanteissa. Näitä alustavia suunnitelmia, eli skeemoja, täydennetään ajoaikana, jotta saataisiin päätettyä, mitä tehdään. Kun halutaan saavuttaa jokin tavoite, suunnittelukerros hakee skeeman, jolla kyseinen tavoite saavutetaan. Skeema sisältää alitavoitteita, jotka saavutetaan toisilla skeemoilla ja niin edelleen.

Vaikka suunnittelukerros yrittää minimoida uudelleensuunnittelua, tulee joskus tilanne, jossa alkuperäisen suunnitelman epäonnistuttua se joutuu muuttamaan olemassaolevia suunnitelmia. Tästä saattaa aiheutua tilanne, jossa tavoitteet ovat ristiriidassa keskenään. Näiden ristiriitatilanteiden välttämiseksi on olemassa mallinnuskerros, joka pyrkii ennustamaan ristiriidat etukäteen ja ehdottamaan uusia tavoitteita, jotka toivottavasti ehkäisevät konfliktien synnyn. Nämä tavoitteet annetaan suunnittelukerrokselle, joka etsii niihin sopivat skeemat.

Koska jokainen kerros on oma itsenäinen prosessinsa, niiden ehdottamat toiminnot voivat olla ristiriidassa keskenään. Tästä syystä niiden syötteet ja tulosteet välitetään kontrollikehyksen kautta. Sen tarkoitus on varmistaa, että agentti käyttäytyy tarkoituksenmukaisesti kaikissa tilanteissa.

Kontrollikehys sisältää joukon niin sanottuja kontrollisääntöjä, joilla se pystyy vaikuttamaan eri kerrosten syötteisiin ja agentin toimintojärjestelmälle annettaviin käskyihin. Kontrollisääntöjä on kahdenlaisia: toiset suodattavat havaintoja ja toiset toimintoja (Kuva 4.4).

---

censor-rule-1:
if entity(obstacle-6) in Perception-Buffer
then
remove-sensory-record(layer-R, entity(obstacle-6))

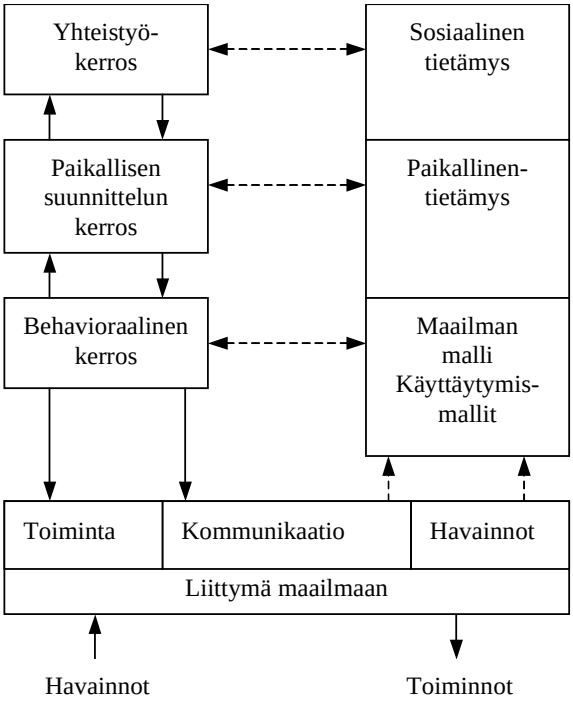
suppressor-rule-3:
if action-command(layer-R-rule-6,
change-orientation-(_)) in Action-Buffer
and
current-intention(start-overtake)
then
remove-action-command(layer-R, change_orientation(_))
and
remove-action-command(layer-M, _)

Kuva 4.4. Kaksi kontrollisääntöä [Ferguson, 1994, 207].

Kuvan 4.4 sääntö censor-rule-1 estää reaktiivista kerrosta havaitsemasta tietynlaista estettä, näin se ei reagoi esteeseen mitenkään. Sääntö suppressor-rule-3 estää reaktiivista kerrosta reagoimasta kaistamerkintään (layer-R-rule-6 estää agenttia ylittämästä tien kaistamerkintöjä) , kun agentti aikoo ohittaa toisen agentin.

4.2.2 INTERRAP

INTERRAP [Müller et al., 1994] on esimerkki kaksisuuntaisesta kerrosarkkitehtuurista. Agentti määritellään joukkona funktionaalisia kerroksia, jotka on linkitetty tietyllä aktivaatiokontrollirakenteella ja jaetulla hierarkkisella tietämyskannalla.



Kuva 4.5. INTERRAP-arkkitehtuuri.

Arkkitehtuuriin kuuluu viisi komponenttia: liittymä maailmaan (world interface, WIF), käyttäytymismalleihin (behaviour) perustuva kerros (behaviour based component, BBC), paikallisen suunnittelun kerros (plan based component, PBC), yhteistyökerros (cooperation component, CC) sekä tietämyskanta. Jokaiselle kerroksella on käytettävissään oma tietämyskannan kerros, johon on kuvattu maailma kyseisen kerroksen tarpeisiin. Tiedon abstraktiotaso kasvaa sitä mukaan, mitä korkeammalle kerrokselle tietämyskannassa edetään.

Behavioraalinen kerros kontrolloi reaktiivista käyttäytymistä ja siihen liittyvää proseduraalista tietämystä tietämyskannassa. Sen toiminta perustuu

käyttäytymismalleihin, joilla agentti suorittaa rutiinitoimintoja ja reagoi muutoksiin ympäristössä ilman että sen tarvitsee suorittaa minkäänlaista symbolista suunnittelua. Paikallisen suunnittelun kerros sisältää nimensä mukaan suunnittelumekanismin, joka tuottaa paikallisia suunnitelmia. Se voi käynnistää alemman kerroksen toimintamalleja saavuttaakseen tiettyjä tavoitteita. Yhteistyökerros vastaa monen agentin järjestelmissä vaadittavasta koordinaatiosta, mitä vaaditaan, kun agenteilla on esimerkiksi yhteisiä suunnitelmia muiden agenttien kanssa jonkin tehtävän hoitamiseksi.

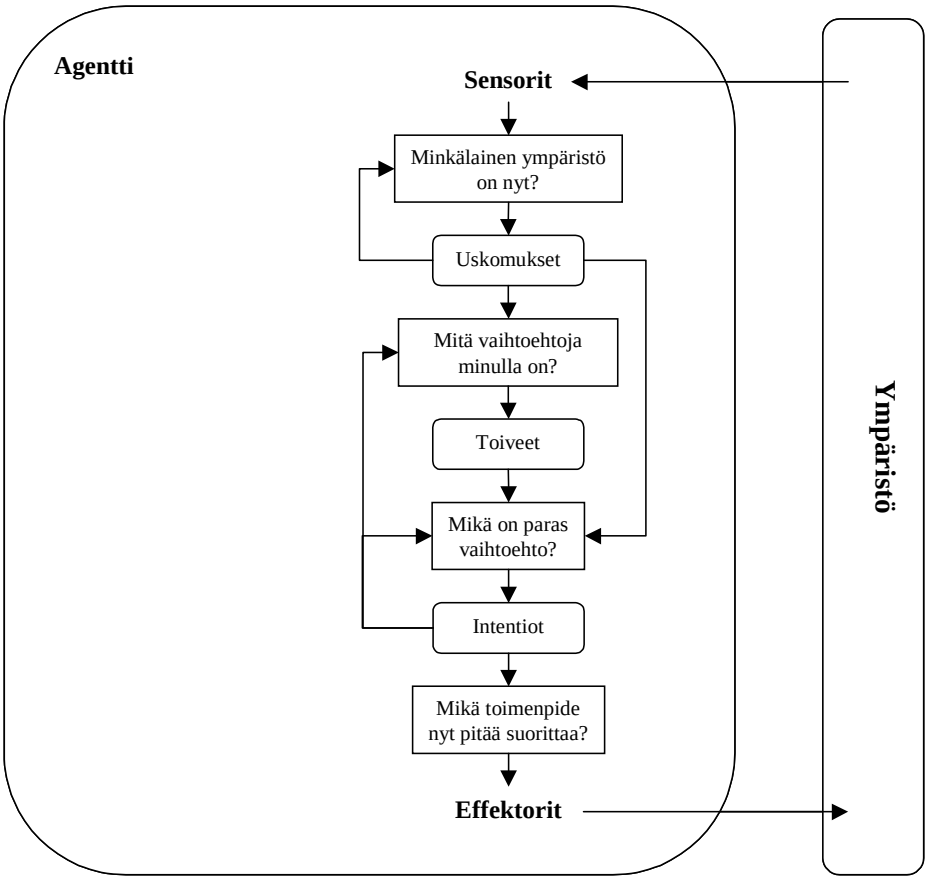
Tietämyskanta on jaettu kerroksiin siten, että alimmalla tasolla talletetaan uskomuksia maailmasta; keskimmäisellä tasolla talletetaan paikallista agentin tilaan liittyvää tietämystä, kuten suunnitelmat, tavoitteet ja aikeet; ylimmällä tasolla talletetaan uskomuksia toisista agenteista, tietoja yhteisistä suunnitelmista, tavoitteista ja aikeista. Kullakin kontrollikerroksella on pääsy vastaavaan tietämyskannan kerrokseen ja sen lisäksi tietämyskannan alempaan kerrokseen.

Jokainen kontrollikerros sisältää kaksi prosessia. *Tilanteen kartoitus ja tavoitteiden aktivointi* (situation recognition and goal activation, SG) tunnistaa kyseistä kerrosta kiinnostavat tilanteet ja aktivoi uusia tavoitteita tuloksena. *Suunnittelu ja skedulointi* (planning and scheduling, PS) ottaa syötteenä saman kerroksen SG:n luomia tilanne-tavoite -pareja ja päättää, millä suunnitelmalla tavoitteet saavutetaan. Lisäksi se skeduloi suunnitelmat suoritukseen ja valvoo niiden suoritusta.

Jos PS-prosessi ei ole pätevä tietyn tilanteen hoitamiseen, se lähettää *aktivaatiopyynnön* (upward activation request) ylemmän kerroksen GS-prosessille. Siellä tilannekuvausta täydennetään ja ratkaistaan uusi tavoite, joka raportoidaan takaisin alemman kerroksen PS-prosessille. Yhteistyökerros ja paikallisen suunnittelun kerros voivat myös lähettää alemmalle kerrokselle käskyn suorittaa jokin toiminto (downward commitment posting). Näin tieto virtaa ylös ja käskyt alas kerrosten välillä.

4.3 BDI-arkkitehtuurit

BDI-arkkitehtuurit perustuvat BDI-malliin ja niissä käytetään käytännön järkeilyä suunnittelussa. Ensimmäinen BDI-arkkitehtuuri oli Intelligent Resource-bounded Machine Architecture (IRMA), joka pyrki toteuttamaan Bratmanin BDI-teorian melko suoraviivaisesti. Tunnetuin BDI-arkkitehtuuri on Procedural Reasoning System (PRS). PRS on muodostunut jonkinlaiseksi referenssitoteutukseksi BDI-agenttien toteutuksessa. Siihen perustuvia arkkitehtuureja ovat mm. dMARS [d'Inverno et al., 1997] ja JAM [Huber, 1999].

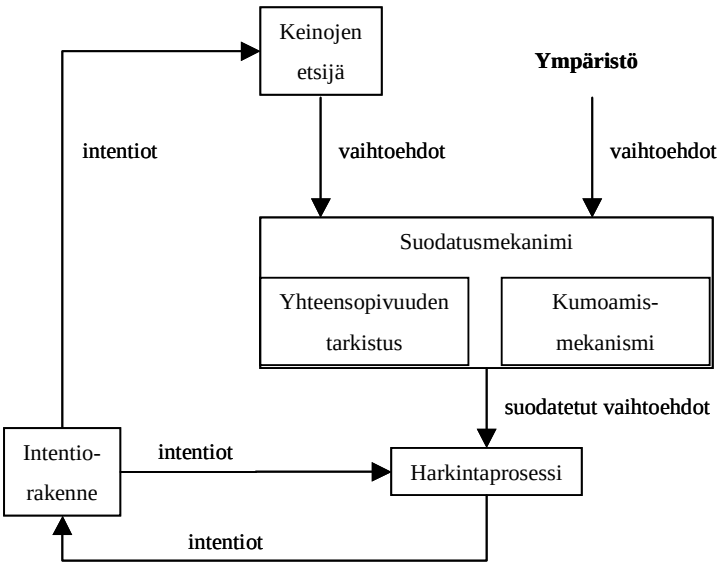


Kuva 4.6. Geneerinen BDI-arkkitehtuuri.

Kuvassa 4.6 on geneerinen BDI-arkkitehtuuri. Siinä agentti ensin päivittää uskomuksensa ympäristön tilaa vastaavaksi. Tämän jälkeen se kartoittaa vaihtoehtonsa ja valitsee niistä parhaan. Lopuksi se päättää, mikä toiminto pitää suorittaa, jotta valittu vaihtoehto saavutettaisiin.

4.3.1 IRMA

Intelligent Resource-Bounded Machine Architecture (IRMA) [Bratman et al., 1988] on käytännön järkeilyä käyttävän resurssirajoitteisen agentin arkkitehtuuri. Se pyrkii toteuttamaan melko suoraviivaisesti Bratmanin BDI-teorian. Kuvassa 4.1 näkyvän IRMA:n intentiot koostuvat puumallisista osittaisista suunnitelmista, joita se on sitoutunut suorittamaan jossain vaiheessa tulevaisuudessa. Näiden suunnitelmien päärooli on toiminnan tuottamisen lisäksi rajoittaa agentin suorittamaa käytännön järkeilyä [Bratman et al., 1988].



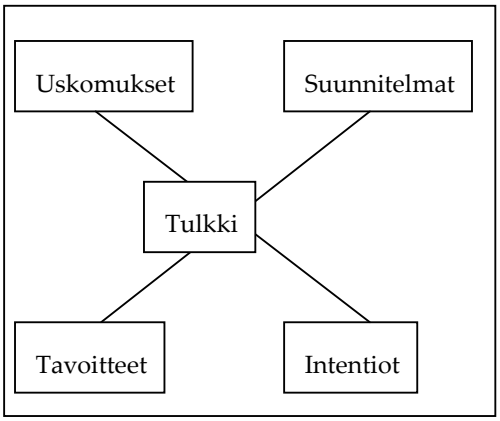
Kuva 4.7. IRMA-arkkitehtuuri [Pollack, 1992].

Arkkitehtuuri sisältää kolme prosessia, jotka osallistuvat päätöksentekoon: keinojen etsijän, suodatusmekanismin sekä harkintaprosessin. Keinojen etsijä esittää suodatusmekanismille vaihtoehtoisia suunnitelmia perustuen suunnitelmakirjastossa oleviin osittaisiin suunnitelmiin. Tilaisuusanalysoija tarkkailee agentin ympäristöä ja esittää myös vaihtoehtoja suodatusmekanismille. Suodatusmekanismi tarkastaa vaihtoehtojen yhteensopivuuden ja antaa suodatuksesta selvinneet vaihtoehdot harkintaprosessille. Suodatusmekanismi sisältää lisäksi sovelluskohtaisen suodatuksenkumoajan, joka saattaa ottaa yhteensopivussuodattimen hylkäämän suunnitelman uudelleenharkintaan, mikäli ympäristössä on tapahtunut tiettyjä muutoksia, jotka laukaisevat kumoamismekanismin.

Harkintaprosessi valitsee suodatetuista suunnitelmista ne, joihin agentti sitoutuu. Suodatuksen idea on vähentää vaihtoehtoisten suunnitelmien määrää, koska harkintaprosessi voi viedä huomattavasti aikaa ja tästä johtuen tuottaa tuloksen, joka on jo syntyessään vanhentunut.

4.3.2 PRS

Kuvassa 4.8 esitetty Procedural Reasoning System (PRS) on tunnetuin BDI-teorian toteuttava agenttiarkkitehtuuri [Wooldridge, 2000]. Se koostuu agentin uskomukset sisältävästä tietokannasta, joukosta tämänhetkisiä saavutettavia tavoitteita, joukosta tavoitteita saavuttavia ja eri tilanteisiin reagoivia suunnitelmia sekä intentiot sisältävästä intentiorakenteesta (intention structure). PRS:n keskeinen komponentti on tulkki. Se valitsee järjetelmän uskomuksiin ja tavoitteisiin sopivia suunnitelmia ja muodostaa niistä intentioita, joita se suorittaa myöhemmin.



Kuva 4.8. PRS-arkkitehtuuri [Georgeff and Ingrand, 1989].

Järjestelmän uskomukset on talletettu tietokantaan. Nämä uskomukset ovat osaksi faktoja järjestelmän staattisista ominaisuuksista, kuten jonkin alijärjestelmän rakenne tai fyysiset lait, joita mekaaniset komponentit noudattavat. Lisäksi tietokannassa on uskomuksia, joita PRS saa suorittaessaan suunnitelmiaan. Tällaisia ovat esimerkiksi havainnot ympäristöstä. Uskomukset ilmaistaan faktoina samaan tapaan kuin esimerkiksi prolog-ohjelmointikielessä [Wooldridge, 2000].

PRS:n *tietämysalueet* (knowledge area) sisältävät tietämystä siitä, kuinka järjestelmän tavoitteita saavutetaan sekä kuinka erilaisiin tilanteisiin reagoidaan. Kukin tietämysalue sisältää rungon ja kutsuehdon. Runko sisältää

tietämysalueen suoritettavat askeleet. Kutsuehto sisältää kuvauksen tilanteesta, jolloin tietämysaluetta voidaan käyttää. Tietämysalueen runkoa voidaan kutsua suunnitelmaksi tai suunnitelman skeemaksi [Georgeff and Ingrand, 1989]. Se on graafi, jossa on yksi alkusolmu ja mahdollisesti monta loppusolmua. Solmujen väliset särmät sisältävät alitavoitteet, joita yritetään tavoittaa suunnitelmaa suoritettaessa. Tietämysalueen onnistunut suorittaminen edellyttää kaikkien alku- ja loppusolmun välillä olevien alitavoitteiden saavuttamista. Joillakin tietämysalueilla ei ole ollenkaan runkoa, vaan niihin on liitetty jokin toiminto, jonka järjestelmä voi suorittaa välittömästi. PRS:llä voi olla myös niinsanottuja metatason tietämysalueita, joilla järjestelmä voi muokata omia uskomuksiaan, tavoitteitaan ja intentioitaan sekä ratkaista eri tietämysalueiden välisiä ristiriitatilanteita. Tällaisten metatason tietämysalueiden lisääminen auttaa huomattavasti agentin päätöksenteossa, mutta niiden suorittaminen ei saa kestää liian kauan, jotta reaktiivisuus ei kärsisi. Tästä syystä niiden suorittamiseen annetaan tiukka aikaraja ja tarvittaessa ne keskeytetään.

Intentiorakenne (intention structure) sisältää ne tehtävät, jotka järjestelmä on valinnut suoritettavaksi joko heti tai tulevaisuudessa [Georgeff and Ingrand, 1989]. Nämä suoritukseen valitut tehtävät muodostavat järjestelmän intentiot. Intentio koostuu alustavasta tietämysalueesta ja mahdollisesti sen alitietämysalueista, jotka täytyy suorittaa, jotta se saadaan suoritetuksi. Tämän ja perinteisen ohjelmointiympäristön prosessin välillä vallitsee suora analogia [Georgeff and Ingrand, 1989]. Intentiorakenne voi sisältää samalla hetkellä monia intentioita, joista osa voi olla keskeytetyssä tilassa ja osa odottaa jonkin ehdon täyttymistä ennen kuin ne voivat jatkaa. Osa voi myös olla metatason intentiota, jotka tekevät päätöksiä järjestelmän sisäisistä uskomuksista, tavoitteista ja intentioista. Intentiot ovat osittain järjestettyjä. Intentio, joka on järjestyksessä aikaisemmin, pitää joko suorittaa tai siitä pitää luopua, ennen kuin järjestyksessä myöhemmin oleva voidaan valita suoritettavaksi.

Koko järjestelmän sydän on *PRS-tulkki* (interpreter). Järjestelmällä on joukko tavoitteita sekä uskomuksia tietyllä ajanhetkellä. Tulkki valitsee näiden tavoitteiden ja uskomusten perusteella alijoukon tietämysalueita. Yksi tai useampi näistä tietämysalueista valitaan suoritukseen ja näin ollen asetetaan intentiorakenteeseen. Tietämysalueen soveltuvuudessa ei käytetä deduktiota, vaan kutsuehtoja verrataan suoraan testaamalla, löytyykö ehtojen muuttujille sopivia arvoja uskomuksista ja tavoitteista. Tämä tekee valinnasta erittäin nopean ja takaa reaktiivisuuden säilymisen [Georgeff and Ingrand, 1989]. Deduktioita voidaan tehdä valitsemalla metatason tietämysalueita suoritettavaksi [Georgeff and Ingrand, 1989]. Kun tietämysalue on valittu, se asetetaan intentiorakenteeseen. Jos kyseessä on uuden tavoitteen saavuttava

tietämysalue, sille luodaan uusi intentio. Jos taas tietämysalue saavuttaa jonkin alitavoitteen, se asetetaan kyseisen intention muodostavan tietämysaluepinon päällimmäiseksi. Lopuksi valitaan jokin intentio suoritettavaksi. Intention seuraava askel sisältää joko suoraan suoritettavan toiminnon tai yhden tai useamman alitavoitteen. Jos kyseessä on toiminto se suoritetaan, alitavoitteet asetetaan järjestelmän tavoitteisiin uusina tavoitteina. Seuraavalla kontrollikierroksella ne laukaisevat uusien tietämysalueiden kutsuehtoja, jotka valitaan suoritukseen ja niin edelleen.

PRS:n tekee kiehtovaksi se, että varsinainen järjestelmä on melko yksinkertainen, mutta metatason tietämysalueilla saadaan agentin kykyjä lisättyä huomattavasti. Tietämysalueiden käyttäminen myös edesauttaa järjestelmien iteratiivista toteutusta. Voidaan ensin toteuttaa prototyyppi alustavilla tietämysalueilla ja sitten tarkentaa ja muuttaa järjestelmää asteittain ilman koko järjestelmän uudestaan kääntämistä.

4.4 Yhteenveto

Agenttiarkkitehtuurit voidaan jakaa löyhästi ryhmiin niiden kontrollimekanismin mukaan. Reaktiiviset arkkitehtuurit toimivat ”tilanne → toiminto” -tyyppisten sääntöjen perusteella. Kerrosarkkitehtuureissa toiminta on jaettu kerroksiin joko horisontaalisesti tai vertikaalisesti. Kerroksia on tyypillisesti kolme, joista alin on reaktiivinen, keskimmäinen proaktiivinen ja ylin sosiaalinen kerros.

BDI-arkkitehtuurit perustuvat käytännön järkeilyyn ja BDI-teoriaan. Niissä agentti punnitsee erilaisia vaihtoehtoja ja sitoutuu joidenkin suorittamiseen. Tämä sitoutuminen vaikuttaa myöhempään päätöksentekoon rajaamalla joitakin vaihtoehtoja pois.

BDI-malli on mielenkiintoinen, koska se perustuu teoriaan ihmisen päättelystä. Siinä oletetaan alusta asti, että agentti on resursseiltaan rajallinen, joten se ei vaadi agenteilta raskaita suunnittelualgoritmeja. Tästä johtuen BDI-agenteissa saadaan yhdistettyä hyvin reaktiivisuus ja proaktiivisuus. Agentti ei vietä liian kauan aikaa suunnittelussa, kun sen täytyy etsiä vain rajallisesta määrästä suunnitelmia. Toisaalta suunnitelmat, joihin agentti sitoutuu, auttavat sitä ennakoimaan tulevaisuuttaan.

5 Monen agentin järjestelmät ja agenttiyhteisöt

Iso osa perinteisestä tekoälystä on keskittynyt siihen, kuinka yksittäinen agentti saadaan toimimaan älykkäästi. Agentit eivät kuitenkaan toimi eristyksissä, vaan ne ovat osa ympäristöä, jossa ne toimivat. Samassa ympäristössä toimii myös toisia agentteja. Syitä hajautettujen agenttijärjestelmien kehittämiseen on monia. Joskus tietojenkäsittely on helpompi suorittaa ja ymmärtää hajautetusti. Kun tietoa on suuri määrä, yhden agentin suunnittelu ja ohjelmointi vaikeutuu ja lisäksi vaarana on, että agentista muodostuu koko järjestelmän pullonkaula. Joskus keskitetty ratkaisu on mahdotonta toteuttaa, koska tieto ja sitä käsittelevät komponentit ovat hajallaan monessa paikassa.

Monen agentin järjestelmissä agentit voivat toimia esimerkiksi älykkäinä sovelluksina, aktiivisina tietolähteinä (vrt. aktiiviset tietokannat) ja tavanomaisia komponentteja ympäröivinä kääreinä (wrapper) [Jennings et al., 1998] . Agentit jakavat tietämystään toisilleen sekä työskentelevät yhteistyössä ja rinnakkaisesti ongelmien parissa. Ne edustavat erilaisia näkökulmia käytettävään tietoon. Agenteilla toteutettu hajautettu järjestelmä ei välttämättä ole tehokkaampi kuin perinteinen hajautettu järjestelmä, mutta koska agentit toimivat autonomisesti, ne lisäävät järjestelmän vikasietoisuutta: Kun yksi komponentti pettää, voivat muut kompensoida tilannetta. Lisäksi agentteja käytettäessä modulaarisuus lisääntyy ja järjestelmien suunnittelu helpottuu.

Agenttijärjestelmällä ja perinteisellä hajautetulla järjestelmällä on kaksi peruseroavaisuutta [Wooldridge, 2002]:

1. Agenttijärjestelmän agentit on voinut suunnitella ja toteuttaa useampi eri henkilö ja niillä voi olla eri tavoitteita. Ne eivät mahdollisesti jaa yhteisiä tavoitteita, vaan niiden täytyy toimia strategisesti, jotta ne saavuttavat tavoittelemansa tuloksen.
2. Koska agenttien pitää tehdä päätöksiä ajoaikana, niiden pitäisi kyetä koordinoimaan toimintansa dynaamisesti. Perinteisissä hajautetuissa järjestelmissä koordinaatio ja yhteistyö on lyöty lukkoon suunnittelussa.

Jennings *et al.* [1998] mukaan monen agentin järjestelmille on ominaista, että yksittäisellä agentilla on riittämättömästi tietoa tai kykyjä ratkaista tiettyjä ongelmia. Lisäksi järjetelmissä ei ole globaalia kontrollia, tieto on hajautettu ja sen käsittely asynronista.

5.1 Agenttien kommunikointi

Yleisesti ottaen agentit eivät voi pakottaa toisia agentteja suorittamaan jotain toimintoa eivätkä muuttaa niiden sisäistä tilaa [Wooldridge, 2002]. Agentit

voivat kuitenkin viestejä lähettämällä suorittaa kommunikatiivisia toimintoja, joilla ne yrittävät vaikuttaa toisten agenttien toimintaan. Tällainen kommunikatiivinen toiminto on esimerkiksi se, jos sanon, että ”ulkona sataa lunta”. Tällöin yritän vaikuttaa kuulijan uskomuksiin. Se, onnistunko siinä, on kuulijasta kiinni.

5.1.1 Puheaktiteoria

Puheaktiteoriassa kommunikaatio nähdään toimintana, kuten mikä tahansa agentin suorittama toiminta. Puheaktiteoria perustuu John Austinin [1962] havaintoon, jonka mukaan osalla luonnollisen kielen lausahduksista on toiminnallisia ominaisuuksia siinä mielessä, että ne muuttavat jollakin lailla maailmaa. Hän kutsui näitä lausahduksia *puheakteiksi*.

Austin listasi joukon performatiivisia verbejä, jotka vastaavat puheakteja. Tällaisia ovat mm. vaatia, luvata ja informoida. Puheaktilla on kolme aspektia:

1. *Lokutio* eli puhujan fyysisesti ääneen lausutut sanat.
2. *Illokutio* eli sanojen tarkoitettu merkitys.
3. *Perlokutio* eli toiminta, joka on illokution tulos.

Jos esimerkiksi opettaja sanoo oppilaalle ”Pyyhi taulu, kiitos”, sisältää akti opettajan ääneen lausutut sanat (lokutio), hänen tarkoituksensa vaatimuksena ikkunan sulkemisesta (illokutio) ja tuloksena todennäköisesti pyyhityn taulun (perlokutio). Ihmisten välisessä kommunikaatiossa viestin tarkoitus ei aina ole selvä, mutta agenttien välillä on viestien sisällön oltava yksiselitteisiä.

Austinin mukaan performatiivisen aktin onnistumisella on kolme ehtoa:

1. On olemassa hyväksytty performatiivia vastaava proseduuri ja olosuhteiden sekä ihmisten tulee olla samat, jotka proseduurissa on määritelty.
2. Proseduuri suoritetaan oikein ja täydellisesti.
3. Aktin tulee olla vilpitön ja kaikki siihen liittyvät toimenpiteet, joihin ryhdytään, tulee suorittaa loppuun, jos mahdollista.

John Searle jatkoi Austinin työtä ja tunnisti useita ominaisuuksia, joiden tulee pitää paikkansa, jotta puhujan ja kuulijan välinen puheakti onnistuisi:

1. Normaalit kuuluvuusolosuhteet.
2. Alustavat ehdot, joiden tulee olla tosia ennen kuin akti voidaan suorittaa. Puhujan tulee osata valita oikea puheakti, puhujan tulee uskoa, että osaa suorittaa toiminnon. Lisäksi pitää päteä ehdon, että kuulija ei suorita toimintoa joka tapauksessa.

3. Vilpittömyysehdot. Puhujan tulee oikeasti haluta, että kuulija suorittaa halutun toiminnon.

Searlen [1979] mukaan illokutioita käytetään viidellä tavalla. *Assertiiveilla* me kerromme ihmisille, kuinka asiat ovat. *Direktiiveillä* yritämme saada heidät tekemään jotain. *Komissiiveilla* sitoudumme tekemään asioita. *Ekspressiiveillä* ilmaisemme tunteitamme ja asenteitamme. Lopuksi, *deklaratiiveilla* ilmoitamme faktoja maailman tilasta.

Selvyyden vuoksi mainittakoon, että kaikki verbit eivät ole performatiiveja. Esimerkiksi sanoma, ”Täten ratkaisen tämän ongelman”, ei luo ratkaisua ongelmaan. ”Vaadin, että ratkaiset tämän ongelman” sisältää perlokution, joka on ongelman ratkaisu edellä mainituin edellytyksin. Performatiiveja käytettäessa sanoman lähettäjän kommunikoima akti on selkeästi määritelty. Viestin tyyppi ei jää epäselväksi vastaanottajalle.

5.1.2 Puheaktit toimintana

Kuten aikaisemmin mainitsin, puheaktit ovat toimintoja, joita agentti suorittaa, kuten mitä tahansa muitakin toimintoja. Puheaktitoiminnon lopputulos ei ole taattu, on vain mahdollista, että jokin asiantila pätee sen jälkeen, kun agentti on suorittanut puheaktinsa. Tämän asiantilan toteutuminen on kiinni siitä, suorittaako kuulija halutun toimenpiteen.

Puheakteille on määritelty erilaisia formalismeja, joiden mukaan niitä käytetään päättelyssä. Esimerkiksi Cohenin ja Perraultin [1980] mukaan agentin informoidessa toista agenttia jostain asiantilasta φ, sen tulee ensiksikin uskoa, että φ. Lisäksi sen tulee uskoa, että se haluaa suorittaa informointiaktin. Suoritetun informointiaktin vaikutuksena on, että kuulija uskoo puhujan uskovan, että φ. Tämä osuus kattaa vasta illokutionaarisen osuuden. Jotta akti onnistuisi, tulee kuulijalla olla jonkinlainen välittävä akti, joka mallintaa aktin perlokutionaarista voimaa. Tällainen akti olla mm. yksinkertainen sääntö, jonka mukaan kuulija uskoo kaiken, mitä puhuja uskoo. Mukaan voidaan lisätä myös ehtoja koskien esimerkiksi puhujan ja kuulijan välistä suhdetta.

Cohenin ja Levesquen [1990b] mallintavat puheaktit toiminnoiksi, joita rationaalinen agentti suorittaa intentioidensa eteenpäinviemiseksi.

5.2 Agenttienväliset viestikielet

Monet tutkijat uskovat, että tehokkaan ja rikkaan *agenttikommunikaatiokielen* (agent communication language, ACL) kehittäminen on avainkysymys agenttiparadigman kehityksessä [Labrou et al., 1999]. Se tarjoaa agenteille tavan vaihtaa informaatiota. Informaation vaihto voidaan toteuttaa erilaisilla

teknologioilla, kuten CORBA ja RMI, mutta agenttikommunikaatiokielet toimivat ylemmällä tasolla niihin nähden. Agenttikommunikaatiokielet käsittelevät propositioita, sääntöjä ja toimintoja semantiikkaa sisältämättömien olioiden sijasta [Labrou et al., 1999]. Lisäksi agenttiviesti kuvaa halutun tilan deklaratiivisella kielellä eikä esimerkiksi proseduuria tai metodia [Labrou et al., 1999].

1990-luvun alkupuolella aloitettiin Yhdysvalloissa DARPA:n (Defense Advanced Research Projects Agency) rahoittama projekti, jonka tarkoituksena oli kehittää tekniikoita, metodologioita ja työkaluja tietämyksen jakamiseen ja uudelleenkäyttöön. Tämän työn tuloksena syntyi ACL-niminen kieli joka sisältää ”sisemmän” kielen, nimeltään KIF (Knowledge Interchange Format), ja ”ulomman” kielen, nimeltään KQML (Knowledge Query and Manipulation Language). KQML määrittelee kielen, jolla agentti voi viestittää toisille haluamansa kommunikatiivisen aktin. KIF on kieli, jolla voidaan esittää tietämystä jostakin tietystä asiasta

5.2.1 KQML

KQML on korkean tason viestipohjainen kieli agenttien väliseen kommunikaatioon. Se perustuu puheaktiteoriaan. KQML sisältää periaatteessa kolme tasoa: sisällön, kommunikaation ja viestin. Sisältötaso sisältää viestin varsinaisen sisällön. KQML voi sisältää minkä tahansa ASCII-merkkijonona tai binäärimuotoisena koodatun sisällön. KQML-toteutus on kiinnostunut sisällön suhteen ainoastaan siitä, mihin se loppuu. Kommunikaatiotaso sisältää alemman tason tietoja, kuten viestin lähettäjä ja vastaanottaja sekä yksilöllinen tunniste.

Viestitaso kertoo, millä verkkoprotokollalla viesti toimitetaan ja minkä performatiivin, eli puheaktin, lähettäjä lisää sisältöön [Labrou et al., 1999]. Performatiivi kertoo viestin tyypin, eli onko kyseessä kysely, vaatimus, ilmoitus tai jokin muu tunnettu performatiivi. Koska viestien sisältöä ei ole määritelty, sisältää viestitaso myös sisältöä kuvailevaa tietoa, kuten sisällön kuvaamisessa käytetyn kielen ja ontologian.

```
(ask-one                                    (tell
    : sender joe                                : sender stock-server
    : content ( PRICE IBM ?price)               : content ( PRICE IBM 14)
    : receiver stock-server                     : receiver joe
    : reply-with ibm-stock                      : in-reply-to ibm-stock
    : language LPROLOG                          : language LPROLOG
    : ontology NYSE-TICKS)                      : ontology NYSE-TICKS)
```

Kuva 5.1 KQML-kysely ja vastaus.

Kuvassa 5.1 vasemmalla palstalla on KQML-viesti joe-nimiseltä agentilta. Viestissä kysytään IBM:n osakkeen hintaa. Tässä viestissä performatiivi on ask-one, joka tarkoittaa, että kysyjä odottaa yhtä vastausta kyselyynsä. Viestin sisältö on PRICE IBM ?price, vastaanottaja on stock-server-niminen agentti, reply-with-parametri kehoittaa käyttämään vastauksesa ibm-stock-tunnistetta, sisällön kieli on LPROLOG ja ontologia NYSE-TICKS. Oikeanpuoleisessa viestissä on vastaus kyselyyn. Performatiivina on tell, eli stock-server kertoo joe-agentille, että lähettäjä on stock-server, sisältö (eli vastaus kyselyyn) on PRICE IBM 14, vastaanottaja on joe, vastaus on kyselyyn ibm-stock, sisällön kieli on LPROLOG ja ontologia NYSE-TICKS.

KQML:ssä on määritelty joukko performatiiveja. Agenttien ei ole pakko käyttää kaikkia niitä ja uusia voi lisätä tarvittaessa. Ainoa rajoitus on, että määriteltyjä performatiiveja tulee käyttää määritelmän mukaisesti. KQML:stä on olemassa monta erilaista murretta, joissa kaikissa on eri määrä performatiiveja. Niitä on murteesta riippuen mainostamiseen, kyselyyn, rekisteröimiseen, ilmoittamiseen, palveluun kirjautumiseen, suosittelemiseen, virheiden käsittelyyn ja niin edelleen.

Alun perin KQML:n semantiikka oli kuvattu informaalisti ja osittaisena. Tämä jätti usein tulkinnan varaa toteutuksessa. Myöhemmin Labrou ja Finin [1997] kehittivät erään murteen performatiiville esiehdot, jälkiehdot ja loppuunsaattamisehdot. Esiehdot ja jälkiehdot määriteltiin sekä lähettäjälle, että vastaanottajalle. Lähettäjän esiehdot kertovat, minkä täytyy päteä, jotta performatiivia voidaan käyttää. Vastaanottajan esiehdot kertovat, minkä täytyy päteä, jotta se voi käsitellä viestin onnistuneesti. Jos vastaanottajan esiehdot eivät täyty, sen todennäköinen vastaus on *error* tai *sorry* [Labrou et al., 1999]. Jälkiehdot kuvaavat lähettäjän tilan performatiivin lähettämisen jälkeen sekä vastaanottajan tilan viestin onnistuneen tulkinnan jälkeen, mutta ennen kuin se on vastannut. Loppuunsaattamisehto kuvaa tilannetta, jossa keskustelun

aikaansaanut intentio on saavutettu onnistuneesti. Esi- ja jälkiehdot eivät takaa, että toiminta onnistuu. Ne kertovat, minkä keskusteluun osallistuvien agenttien tilan oletetaan oletetaan olevan.

5.2.2 FIPA ACL

The Foundation for Intelligent Physical Agents (FIPA) on vuonna 1996 perustettu kansainvälinen organisaatio, jonka tarkoitus on edistää älykkäiden agenttien käyttöä kehittämällä moniagenttijärjestelmiä tukevia spesifikaatioita.

Osittain KQML:n saamasta kritiikistä johtuen FIPA on määritellyt FIPA ACL-agenttikommunikaatiokielen. FIPA ACL on päällisin puolin hyvin samanlainen KQML:n kanssa: se on ”ulompi” kieli viesteille eikä sekään oleta mitään tiettyä kieltä viestin sisällölle.

```
(query-ref                          (inform
    : sender joe                        : sender stock-server
    : content ( PRICE IBM ?price)       : content ( PRICE IBM 14)
    : receiver stock-server             : receiver joe
    : reply-with ibm-stock              : in-reply-to ibm-stock
    : language LPROLOG                  : language LPROLOG
    : ontology NYSE-TICKS)              : ontology NYSE-TICKS)
```

Kuva 5.2 FIPA ACL -kysely ja vastaus

Kuvassa 5.2 on esitetty kuvan 5.1 KQML-viestejä vastaavat FIPA ACL-viestit. Kuten havaitaan, viestit ovat hyvin samankaltaisia; vain performatiivit, joita FIPA ACL:ssä kutsutaan kommunikatiivisiksi akteiksi, ovat muuttuneet. FIPA ACL ja KQML ovat kuitenkin erilaiset semantiikaltaan. Kun KQML ei aluksi sisältänyt formaaleja määritelmiä performatiiveilleen, niin FIPA ACL:ssä ne sisällytettiin mukaan alusta alkaen. FIPA ACL:n performatiivit on esitetty kuvassa 5.3. Se sisältää kommunikatiivisia akteja tiedon välittämiseen ja vaatimiseen, neuvotteluun, toimintojen suorittamiseen ja virheiden käsittelyyn.

Performatiivi	Informaation välitys	Informaation vaatiminen	Neuvottelu	Toimintojen suorittaminen	Virheiden käsittely
accept-proposal			x		
agree				x	
cancel		x		x	
cfp			x		
confirm	x				
disconfirm	x				
failure					x
inform	x				
inform-if	x				
inform-ref	x				
not-understood					x
propagate				x	
propose			x		
proxy				x	
query-if		x			
query-ref		x			
refuse				x	
reject-proposal			x		
request				x	
request-when				x	
request-whenever				x	
subscribe		x			

Kuva 5.3. FIPA ACL:n kommunikatiiviset aktit.

FIPA ACL -spesifikaatio koostuu joukosta viestityyppejä ja kuvauksista niiden käytöstä. Kuvaukset sisältävät sanallisen osan ja formaalilla modaalilogiikkaan perustuvalla SL (Semantic Language) -kielellä laaditun osan [FIPA, 2000b]. SL-kieli sisältää modaaliset operaattorit uskomuksille (B), toiveille (D), epävarmoille uskomuksille (U) ja intentioille, jotka kuvataan pysyvinä tavoitteina (persistent goal, PG). SL-kieli kykenee ilmaisemaan propositioita, olioita ja toimintoja. FIPA ACL:ssä jokaisen kommunikatiivisen aktin semantiikka määritellään joukkona SL-kaavoja, jotka kuvaavat aktin *toteutettavuusehdot* sekä *rationaalisen vaikutuksen*. Kommunikatiivisen aktin toteutettavuusehto kuvaa ehdot, jotka lähettäjän tilan on täytettävä, jotta se voisi käyttää aktia. Rationaalinen vaikutus kuvaa, vaikutusta, jonka agentti voi olettaa aktilla olevan. Se määrittää yleensä myös ehdot, jotka ovat oltava tosia vastaanottajalle. Vastaanottavaa agenttia ei velvoiteta pitämään huolta, että odotettu vaikutus tapahtuu. Se voi esimerkiksi epäonnistua tai havaita vaaditun toiminnan mahdottomaksi. Näin ollen agentti voi käyttää tietämystään rationaalisesta vaikutuksesta suunnitellakseen, mitä aktia se käyttää, mutta se ei voi olettaa, että rationaalinen vaikutus seuraa vääjäämättä aktin suorittamisesta [Labrou et al., 1999].

5.3 Ontologiat

Jos kaksi agenttia haluaa kommunikoida keskenään jostakin, on niiden sovittava käytettävästä terminologiasta, jolla kyseistä ilmiötä kuvataan. Tällaista käsitteitä kuvaavaa terminologiaa kutsutaan ontologiaksi. Ontologia määrittää tietokoneen ymmärtämässä muodossa kuvattavan sovellusalueen käsitteet ja niiden väliset suhteet. Joskus puhutaan myös taksonomiasta ontologiana, mutta taksonomia ei pelkästään riitä, vaan ontologia kuvaa käsite-alikäsite-suhteen lisäksi myös muita suhteita.

Noyn ja McQuinessin [2001] mukaan hyvä ontologia helpottaa sovellusalueen tietämyksen uudelleenkäyttöä ja auttaa analysoimaan sitä. Lisäksi heidän mukaansa ontologioita käytetäessä sovellusalueen tietämys tulee erotetuksi operationaalisesta tietämyksestä.

Guarino [1997] jakaa ontologiat niiden yksityiskohtaisuuden ja tehtävästä riippuvuuden perusteella karkeasti neljään luokkaan: Ylimmän tason ontologioihin, sovellusalueontologioihin, tehtäväontologioihin ja sovellusontologioihin. Ylimmän tason ontologiat kuvaavat erittäin yleisellä tasolla olevia käsitteitä, kuten tila, aika, objekti, tapahtuma ja toiminto. Sovellusalueontologiat ja tehtäväontologiat kuvaavat geneeriseen sovellusalueeseen (kuten lääketiede tai autot) tai tehtävään (kuten diagnosointi tai myynti) liittyvää sanastoa erikoistamalla ylimmän tason käsitteitä. Sovellusontologiat erikoistavat sovellusalueontologian käsitteitä. Ne kuvaavat niitä käsitteitä (kuten korvattava yksikkö tai varaosa), jota sovelluksen toimijat käyttävät, kun ne suorittavat aktiviteettejaan.

5.3.1 KIF

KIF on tarkoitettu KQML-viestien sisältöosan kuvaamiseen. KIF-kieli perustuu ensimmäisen kertaluokan predikaattilogiikkaan ja on näinollen erittäin ilmaisuvoimainen. Sillä voi kuvata sovellusalueen käsitteiden ominaisuuksia ja niiden välisiä suhteita sekä sovellusalueen yleisiä ominaisuuksia [Wooldridge, 2002].

KIF-kieltä ei tarkoitettu alunperin ihmisten prosessoitavaksi, mutta sille on kehitetty työkaluja, joilla ihmiset ovat tuottaneet ontologioita. Eräs KIF:n sovellus on Ontolingua [Farquhar et al., 1996]. Ontolingua-palvelin on www-pohjainen palvelu, jonka avulla eri ryhmät voivat jakaa ontologioita toistensa käyttöön. Ontologiat on kuvattu KIF-kieleen perustuvalla Ontolingua-ontologiakielellä.

5.3.2 RDF, DAML ja OWL

Resource Description Framework (RDF) on mekanismi, jolla nimensä mukaisesti kuvataan erilaisia resursseja. Se on suunniteltu tietokoneen ymmärrettävissä olevan tiedon välitykseen webissä. Sen määritteleminen aloitettiin World Wide Web Consortiumissa (W3C) samoihin aikoihin XML:n kanssa.

RDF:n käyttämä tietomalli perustuu lausekkeisiin (statement), jotka ovat muotoa {predikaatti, subjekti, objekti} [Lassila and Swick, 1999]. Esimerkiksi lausekke {”kirjoittaja”, [http://foo.bar.com/gradu.doc], ”Janne”} kertoo, että resurssilla nimeltä http://foo.bar.com/gradu.doc on ominaisuus (property), nimeltä ”kirjoittaja” ja tämän kirjoittajan arvo on ”Janne”.

RDF koodataan useimmiten XML-muodossa, jolloin edellinen lauseke on muodossa:

```
<rdf:Description about=”http://foo.bar.com/gradu.doc”>
    <a:kirjoittaja>Janne</a:kirjoittaja>
</rdf:Description>
```

RDF sisältää oliopohjaisten kielten tapaan luokkajärjestelmän, jolla kuvattavia resursseja sekä ominaisuuksia voidaan tyypittää[Lassila and Swick, 1999]. Tällainen tyypittäminen tehdään RDF-skeema-spesifikaatiokielellä [Brikley and Guha, 2003], joka on itsessään kuvattu RDF:llä. RDF-skeema-kielellä tehtyjä tyypityksiä kutsutaan sanastoiksi. Yhdessä RDF-kuvauksessa voi olla käsitteitä monesta sanastosta.

Darpa Agent Markup Language (DAML) perustuu XML:n merkintätapaan ja RDF:n lausekkeisiin. DAML laajentaa RDF:n luokkakäsitettä ja mahdollistaa RDF-skeemakieleen verrattuna rikkaampien kuvausten tuottamisen. Myöhemmin mukaan on otettu vielä Ontology Inference Layer (OIL) -kielen frame-pohjaisia käsitteitä. Tuloksena on erittäin ilmaisuvoimainen ontologioiden kuvaamiseen käytety kieli.

W3C on kirjoitushetkellä määrittelemässä uutta RDF-pohjaista ja DML+OIL -kielestä johdettua Web Ontology Language (OWL) -ontologiakieltä. OWL [Dean and Schreiber, 2003] on tarkoitettu ontologioiden julkaisemiseen ja jakamiseen webissä. OWL-dokumentti sisältää vapaaehtoisia ontologiaotsikoita (yleensä korkeintaan yksi), luokka- ja ominaisuusmäärittelyjä sekä yksilöitä (luokan ilmentymä) kuvaavia faktoja [Dean and Screiber, 2003].

5.4 Agenttien koordinointi

Agenttien koordinointi on moniagenttijärjestelmässä erittäin tärkeää. Ilman sitä kaikki kommunikaation tuomat edut katoavat ja jäljelle jää kaoottinen joukko

itsenäisesti toimivia agentteja. On useita syitä, miksi agenttien toiminta tulee koordinoida [Jennings, 1996; Nwana et al., 1996]:

1. Jotta estettäisiin kaaos. Ilman koordinaatiota agenttipohjainen järjestelmä ajautuu helposti anarkiaan, koska agenteilla on usein vain lokaali näkökulma järjestelmään.
2. Jotta järjestelmän globaaleissa rajoitteissa pysyttäisiin.
3. Järjestelmän agenteilla voi olla erilaisia taitoja. Ne pitää koordinoida samaan tapaan kuten esimerkiksi lääkärien, sairaanhoitajien ja ambulanssin kuljettajan toiminta potilasta hoidettaessa.
4. Eri agenttien tavoitteet ja toiminnot ovat usein toisistaan riippuvia.
5. Tehokkuus. Kun yhden agentin havaitsema tieto on käyttökelpoista myös toiselle agentille, ne voivat ratkaista yhdessä ongelman nopeammin.

Yleensä agenttien täytyy kommunikoida keskenään, jotta ne saataisiin koordinoitua, mutta ei aina. Joskus toiminnot koordinoidaan ylläpitämällä mallia toisista agenteista, niiden uskomuksista ja aikomuksista. Jos oletetaan, että toiset agentit jakavat saman näkemyksen ympäristöstä, voidaan esimerkiksi peliteoreettisella analyysillä päätellä, miten missäkin tilanteessa tulee toimia [Wooldridge, 2002].

5.4.1 Organisaation rakenne

Yksinkertaisin koordinaatiotekniikka hyödyntää etukäteen mallinnettua organisaation rakennetta. Organisaatio voi perustua esimerkiksi hierarkiaan, asiantuntijayhteisöön, markkinoihin tai tieteelliseen yhteisöön [Sycara, 1998].

Hierarkkisessa organisaatiossa agentit toimivat isäntä/orja-tyyppisesti, jolloin isäntä jakaa tehtäviä ja resursseja orjilleen. Asiantuntijayhteisössä keskenään tasavertaiset eri alojen asiantuntijat ratkovat yhdessä ongelmia. Markkinoilla agentit kilpailevat keskenään tehtävistä tai resursseista. Tieteellisessä yhteisössä agentit ratkovat ongelmat lokaalisti ja jakavat tuloksen muiden tarkistettavaksi.

5.4.2 Urakointi – Contract Net

Urakointipohjaisessa koordinoinnissa tehtäviä ja resursseja jaetaan erityisellä urakkaprotokollalla. Contract net -protokolla on klassinen esimerkki tällaisesta protokollasta. Se on korkean tason protokolla, jota käyttämällä voidaan jakaa

tehtäviä. Contract net on käyttökelpoinen, kun tehtävä tai ongelma on luonteeltaan hierarkkinen ja jaettavissa karkeasti osiin. Sitä voidaan käyttää erilaiseen organisaatioon perustuvissa järjestelmissä.

Contract net -protokollassa agenteilla voi olla kaksi roolia [Nwana et al., 1996]:

1. Manageri jakaa tehtävän alitehtäviksi ja hakee urakoitsijoita suorittamaan niitä.
2. Urakoitsija suorittaa alitehtävän. Se voi myös omaksua managerin roolin ja jakaa saamansa tehtävän alitehtäviksi ja antaa ne aliurakoitsijoiden suoritettaviksi.

Protokolla alkaa, kun manageri lähettää tarjouspyynnön suoritettavista tehtävistä potentiaalisille urakoitsijoille. Urakoitsijat lähettävät managerille urakkatarjouksen annettuun takarajaan mennessä. Manageri valitsee parhaan tarjouksen ja lähettää tehtävän sen tehneelle urakoitsijalle. Lopuksi urakoitsija lähettää tuloksensa managerille.

5.4.3 Moniagenttisuunnittelu

Moniagenttisuunnittelussa (multiagent planning) agentit muodostavat yhteisen suunnitelman, joka koordinoi niiden toiminnan. Moniagenttisuunnittelu voi olla keskitettyä tai hajautettua [Nwana et al., 1996].

Keskitetyssä moniagenttisuunnittelussa agentit toimittavat osittaiset suunnitelmansa agentille, joka ratkoo suunnitelmien väliset ristiriidat ja kokoaa niistä yhteisen suunnitelman. Hajautetussa moniagenttisuunnittelussa agentit muodostavat ensin omat tavoitteensa ja generoivat lyhyen aikavälin suunnitelmat niiden saavuttamiseksi. Sen jälkeen ne jakavat suunnitelmansa ja tavoitteensa muiden agenttien kanssa nähdäkseen, missä kohdin ne ovat vuorovaikutuksessa keskenään. Lopuksi agentit muuttavat omia suunnitelmiaan, jotta niiden omat aktiviteetit tulisivat paremmin koordinoiduiksi.

5.4.4 Jaetut intentiot

Intentiot, joista puhuttiin luvussa kolme, ovat tärkeitä yhteistyön koordinoinnissa. Ne tarjoavat sekä vakautta, että ennustettavuutta, joita tarvitaan sosiaalisessa kanssakäymisessä: Niin oikeassa elämässä, kuin agenttienvälisessä kanssakäymisessäkin. Ne tuovat myös joustavuutta ja reaktiivisuutta, joita tarvitaan, kun toimitaan muuttuvassa ympäristössä [Wooldridge, 2002].

Wooldridgen [2002, s. 204] esimerkkiä mukaillen tieto, että tämän kirjoitushetkellä olen jo pitänyt kesäloman, voi auttaa ystäviäni koordinoimaan

suunnitelmiaan. Esimerkiksi Espanjan-matka minun kanssani ei tällä hetkellä ole heille mahdollinen suunnitelma.

On tärkeää erottaa koordinoitu yhteistyö koordinoidusta toiminnasta. Kun ihmisjoukko juoksee sadekuuron sattuessa saman puun alle, ei ole kyse yhteistoiminnasta, vaikka toiminta on koordinoitua. Jos sama tapahtuu esimerkiksi osana tanssin koreografiaa, on kyse yhteistyöstä.

Kuuluminen ryhmään aiheuttaa ihmisillä jonkinlaista vastuuta ryhmän toisten jäsenien suhteen. Vaikka aikomuksen jättäminen saattaisi olla subjektiivisesti ottaen rationaalinen ratkaisu, voi sitoutuminen ryhmään ajaa ratkaisun ohi. Jos yksi raskasta taakkaa kantavasta ryhmästä toteaa, että taakka on liian painava, olisi periaatteessa järkevää vain pudottaa taakka. Mutta koska työtä on tekemässä useampi, on järkevää ainakin ilmoittaa muille aikomuksestaan pudottaa taakka.

5.4.5 Sosiaaliset konventiot

Kun ryhmä agentteja ryhtyy toimimaan yhteistyössä, ne sitoutuvat sekä ryhmän yhteiseen tavoitteeseen, että niille annettuihin omiin tehtäviin. Sekä yhteiset, että omat sitoumukset ovat pysyviä. Toisin sanoen, niistä pidetään kiinni, kunnes ne käyvät tarpeettomiksi. Sen, milloin yhteisistä sitoumuksista voidaan luopua, määräävät ns. sosiaaliset konventiot. Konventiot kuvaavat myös, kuinka agentin tulee käyttäytyä muita ryhmän jäseniä kohtaan. Agentilla voi olla esimerkiksi konventio, jonka mukaan sen on ilmoitettava muille, jos se luopuu yhteisestä sitoumuksesta. Sosiaalisia konventioita käytettäessä agentti tietää, mitä muut odottavat siltä ja myös mitä kaikilta muilta odotetaan.

Konventiot tuovat sosiaalisia rajoitteita. Ne tasapainottavat yksilöllistä vapautta ja toisaalta agenttiyhteisön yhteisiä tavoitteita sekä helpottavat huomattavasti päätöksentekoprosessia kertomalla tarkasti, mitä pitää tehdä tietyissä tilanteissa [Wooldridge, 2002]. Agenttiyhteisön konventiot joko suunnitellaan etukäteen tai ne muotoutuvat itsestään järjestelmän toimiessa. Ensin mainittu on helpompi toteuttaa ja antaa suunnittelijalle paremman mahdollisuuden kontrolloida järjestelmän toimintaa kokonaisuutena. Wooldridgen [2002] mukaan konventioita ei voi suunnitella etukäteen, jos järjestelmän ominaisuuksia ei tiedetä suunnitteluhetkellä tai jos monimutkaisen järjestelmän agenttien tavoitteet muuttuvat koko ajan. Tällöin olisi hyödyllistä, jos agentit pystyisivät keskenään sopimaan konventioista.

Ongelmat konventioiden sopimisessa dynaamisesti ovat ilmeisiä: Miten agentit voivat päästä sopimukseen globaaleista sosiaalisista konventioista käyttäen vain lokaalia informaatiota? Kaikkien agenttien pitäisi noudattaa

konventioita, mutta jokaisen on päätettävä itse paikallisesti, mitä niistä se alkaa noudattaa.

Lähdekirjallisuuden terminologia vaihtelee tässä kohdin. Niissä puhutaan konventioista, normeista ja sosiaalisista normeista. Konventio on sosiaalinen tapa, joka ei ole lain eikä moraalin sanelema [Aikio ja Vornanen, 1992]. Normi on ryhmälle tyypillinen käyttäytymistapa tai piirre [Aikio ja Vornanen, 1992]. Sosiaalinen normi on käyttäytymissääntö, jonka rikkomisesta seuraa rangaistus [Aiko ja Vornanen, 1992]. Näillä kaikilla kuitenkin tarkoitetaan, että yhteisöön kuuluminen säätelee yksilön toimintaa jollakin tavalla.

5.4.6 Neuvottelu

Suurin osa koordinaatiomenetelmistä käyttää jonkinlaista neuvottelua [Nwana et al., 1996]. Bussmannin ja Mullerin [1992] mukaan neuvottelu on kommunikaatioprosessi, jolla ryhmä agentteja saavuttaa yhteisesti hyväksytyn sopimuksen koskien jotain asiaa. Jokainen neuvottelu sisältää [Wooldridge, 2002]:

1. neuvottelujoukon, joka kuvaa mahdolliset ehdotukset, joita agentti voi tehdä,
2. protokollan, joka määrittelee lailliset ehdotukset, joita agentti voi tehdä neuvotteluhistorian perusteella,
3. kokoelman agenttien yksityisiä strategioita, jotka kuvaavat, mitä propositioita agentit tulevat tekemään, sekä
4. säännön, joka määrittelee, milloin on saavutettu sopimus ja minkälainen se on.

Neuvottelussa agentit tekevät jokaisella neuvottelukierroksella oman ehdotuksensa. Ehdotukset määritellään agentin strategiassa, valitaan neuvottelujoukosta ja ovat protokollan mukaisia [Wooldridge, 2002]. Jos sopimussäännön määrittelemä ehto täyttyy, neuvottelu päättyy ja sopimus tehdään.

5.4.7 Interaktioprotokollat

Useimmissa koordinaatiotekniikoissa tarvitaan kommunikaatiota. Agenttienvälinen kommunikaatio etenee melko kaavamaisesti. Siinä lähetetään ja vastaanotetaan tiettyjä samalla lailla toistuvia viestejä. Tällaista kommunikaatiomallia kutsutaan agenttien keskusteluksi, eli interaktioprotokollaksi. Viesteinä lähetettävät puheaktit tulkitaan keskustelun kontekstissa.

Interaktioprotokollien semantiikka on kolmella tasolla [Pitt and Mamdani, 1999]:

1. sisältötasolla, jolla tulkitaan viestien sisältö,
2. toimenpidetasolla, jolla päätetään, millä puheaktilla mihinkin viestiin vastataan, ja
3. intentionaalisella tasolla, jolla päätetään kommunikaation aloittamisesta yleensä.

Tasot yksi ja kaksi ovat agentin kannalta sisäisiä ja kolme ulkoinen. Interaktioprotokollat tulisi määritellä tasolla kaksi, jolloin niitä voidaan käyttää monilla erilaisilla agenteilla.

Tyypillisiä esimerkkejä interaktioprotokollista ovat edellä mainittu Contract Net -protokolla sekä erilaiset huutokauppa- ja äänestysprotokollat.

5.5 Yhteenveto

Monen agentin järjestelmissä agentit lähettävät toisilleen viestejä ja käyvät niiden avulla yksinkertaisia keskusteluja. Viestit perustuvat puheaktiteoriaan, joka kuvaa kommunikaation toimintana. Teorian mukaan puhujan voi muuttaa maailmaa tietyillä verbeillä, joita kutsutaan performatiivisiksi verbeiksi.

Agenttien koordinointi on tärkeää moniagenttijärjestelmissä. Koordinaatio voi perustua erilaisiin protokolliin, organisaatioon, yhteiseen suunnitteluun, jaettuihin intentioihin, sosiaalisiin konventioihin sekä neuvotteluun.

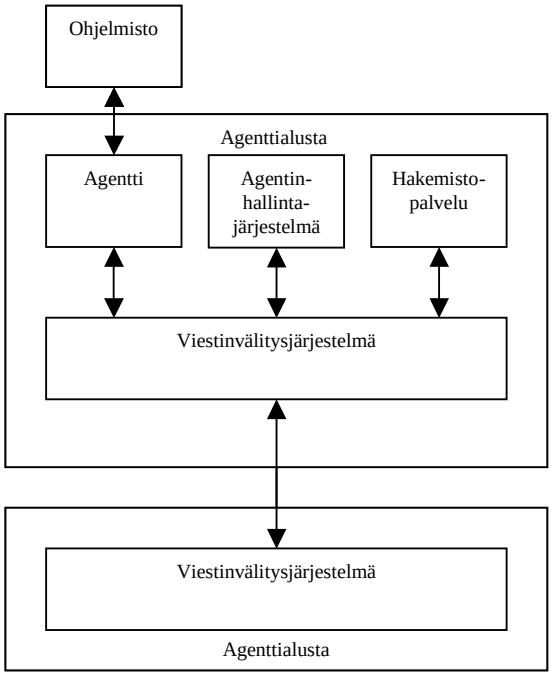
6 FIPA-yhteensopivan agenttiympäristön arkkitehtuuri

Tässä luvussa esittelen tarkemmin FIPA:n (The Foundation for Intelligent Physical Agents) tekemää työtä agenttipalveluiden määrittelyssä sekä erään FIPA-pohjaisen agenttiympäristön, jossa käytetään Enterprise Java Bean -komponentteja FIPA:n määrittelemien palveluiden toteutuksessa.

6.1 FIPA

The Foundation for Intelligent Physical Agents (FIPA) on vuonna 1996 perustettu kansainvälinen organisaatio, jonka tarkoitus on edistää älykkäiden agenttien käyttöä kehittämällä moniagenttijärjestelmiä tukevia spesifikaatioita. FIPA:n jäseninä on agentteja tutkivia ja kehittäviä yrityksiä ja yliopistoja. Jäsenet ovat kollektiivisesti sitoutuneet avoimuuteen agenttipohjaisten sovellusten ja palveluiden kehittämisessä. Tuotetut spesifikaatiot ovat avoimia ja kaikkien käytettävissä.

FIPA aloitti vuonna 1996 määrittelemällä FIPA ACL -kommunikaatiokielen. Myöhemmin määrittely kohdistui FIPA-agenttialustaan, sen palveluihin ja agenttien elinkaareen.



Kuva 6.1. Agenttienhallinan referenssimalli [FIPA, 2000a].

6.1.1 Agenttialusta

FIPA-agentit sijaitsevat fyysisesti *agenttialustalla* (Agent Platform, AP) ja käyttävät hyväkseen sen tarjoamia palveluita. FIPA:n määrittelemät agenttialustan pakolliset palvelut ovat *agentinhallintajärjestelmä* (Agent Management System, AMS), yksi tai useampi *palveluhakemisto* (Directory Facilitator, DF) sekä *viestinvälityspalvelu* (Message Transport Service, MTS). Agenttialustan palvelut ovat loogisia kokonaisuuksia, ne voivat olla yhden komponentin palveluja tai erillisä komponentteja. Se, miten ne toteutetaan, on jätetty toteuttajan harkittavaksi.

Agentit voivat rekisteröidä palveluitansa ja lähettää hakuja toisten palveluista palveluhakemistoon. Jos palveluhakemistossa ei ole hakupyyntöä vastaavaa palvelua, se voi lähettää haun edelleen toisiin palveluhakemistoihin. Agenttienhallintajärjestelmä kontrolloi agenttialustan käyttöä ja pääsyä siihen. Agentit rekisteröivät agenttitunnisteensa (agent identifier) ja osoitteensa agentinhallintajärjestelmään. Siitä voi myös hakea toisten agenttien osoitteita.

Viestinvälityspalvelu on oletusarvoinen viestikanava eri agenttialustoilla sijaitsevien agenttien välillä. Alustan sisäisessä viestinvälityksessä voidaan käyttää mitä tahansa viestijärjestelmää. FIPA:n spesifikaatioissa keskitytään eri agenttialustoilla olevien agenttien väliseen yhteistoimintaan.

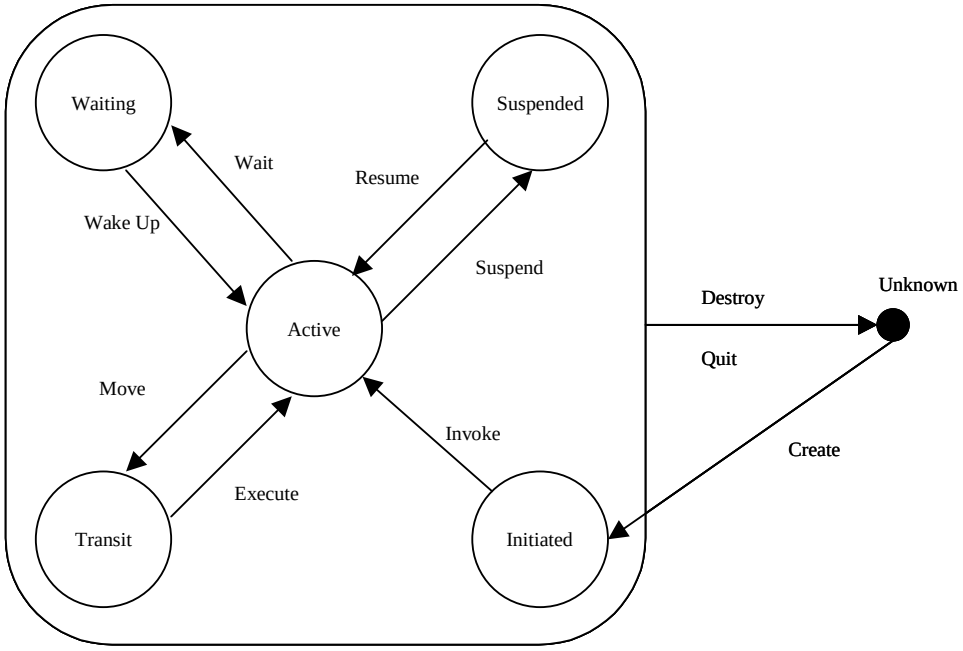
6.1.2 FIPA-agentti

Agenttialustan perustoimija on agentti. Agentti tarjoaa yhden tai useamman palvelun. Palvelu voi olla esimerkiksi yhteys alustan ulkopuolisiin ohjelmiin, ihmisiin tai kommunikaatiopalveluihin. FIPA-agentilla on yksikäsitteinen tunniste, joka erottaa ne muista FIPA-agenteista sekä ainakin yksi omistaja. Lisäksi sillä on joukko osoitteita, joihin sille voi lähettää viestejä. Agentti rekisteröi osoitteensa agenttienhallintajärjestelmään.

Vaikka FIPA ei ota kantaa agenttien tai agenttialustan toteutukseen, on se määritellyt kuitenkin agentin mahdolliset tilat alustalla. Nämä näkyvät kuvassa 5.2. Agentin tilasta riippuu, miten viestinvälityspalvelu käsittelee agentille suunnattuja viestejä.

Kun agentti luodaan Create-komennolla, se siirtyy Initiated-tilaan. Tällöin agentti ei vielä ole toiminnassa, vaan se aktivoidaan Invoke-komennolla, joka siirtää sen Active-tilaan. Tällöin viestejä välitetään agentille normaalisti. Agentti voi siirtyä odottamaan jotain tapahtumaa Wait-komennolla. Waiting-tilasta palaaminen tapahtuu agenttialustan suorittamalla Wake Up -komennolla. Suspend-komento siirtää agentin Suspended-tilaan, jolloin agentti on pysähtynyt. Agenttialustan suorittama Resume-komento palauttaa sen Active-tilaan. Jos alusta tukee liikkuvia agentteja, asetetaan agentti Move-komennolla

Transit-tilaan, jolloin agentti voidaan siirtää määränpäähänsä. Siirtymisen jälkeen se palautetaan Execute-komennolla takaisin Active-tilaan.



Kuva 6.2. Agentin elinkaari agenttialustalla[FIPA, 2000a].

Waiting-, Suspended- ja Transit-tilassa viestinvälityspalvelu puskuroi agentin viestit odottamaan agentin paluuta Active-tilaan. Destroy lopettaa agentin voimaperäisesti. Quit-komento sammuttaa agentin ”armollisesti”, jolloin se voi suorittaa tarvittavat lopetustoimet ennen toiminnan loppumista. Agentti voi kieltäytyä Quit -komennosta, mutta ei Destroy-komennosta. Kumpikin komento siirtää agentin Unknown-tilaan, jolloin myös viestien välittäminen lopetetaan.

6.1.3 Agenttien nimeämimen

Jokaisella FIPA-agentilla on oma yksilöllinen agenttitunniste (Agent Identifier, AID). Tämä tunniste koostuu agentin nimestä sekä parametreista, kuten viestinvälitysosoitteista ja nimipalveluiden osoitteista.

Agentin nimi on pakollinen, parametrit ovat vapaaehtoisia. FIPA [2000a] määrittelee kaksi parametria, *osoitteet* ja *nimenselvittäjät* (resolvers). Osoitteet-parametri sisältää järjestetyn listan osoitteita, joihin agentille voi lähettää

viestejä. Jokaiselle viestinvälitysprotokollalle on oma osoitteensa. Agentilla voi olla esimerkiksi HTTP-protokollaa käyttävä osoite muotoa http://foo.com/agent ja IIOP-protokollaa käyttävä osoite muotoa iiop://foo.com/agent. Nimenselvittäjät-parametri sisältää järjestetyn listan sellaisten palveluiden osoitteita, joilta voi tiedustella agentin osoitetta sen nimen perusteella. Yleensä AMS tarjoaa tällaista nimipalvelua, mutta mikään ei estä muitakin agentteja tarjoamasta sitä. Kumpikin lista on järjestetty siten, että ne osoitteet, joita agentti toivoo käytettävän, on asetettu listan alkuun.

Agentin parametreja voi muuttaa ja lisätä luomisen jälkeen, mutta sen nimi pysyy muuttumattomana koko elinkaaren ajan. Agenteille voi lisätä myös parametreja, jotka eivät ole FIPA:n määrittelemiä, kuten lempinimi tai rooli organisaatiossa [FIPA, 2000a]. Agenttitunnisteen nimiosa on globaalisti yksiselitteinen. Yksinkertainen tapa muodostaa nimi on koota se agentin varsinaisesta nimestä ja kotialustan (Home Agent Platform, HAP) nimestä '@' -merkillä yhdistettynä sähköpostiosoitteiden tapaan, esimerkiksi agentti@foo.com.

6.1.4 Agenttienhallintajärjestelmä

Agentinhallintajärjestelmä, (Agent Management System, AMS), on pakollinen osa agenttialustaa. Jokaisella agenttialustalla on yksi AMS. AMS:n vastuulla ovat agenttialustan agentteihin liittyvät toiminnot, kuten agenttien luonti ja tuhoaminen, agenttien dynaamisesta rekisteröinnistä päättäminen sekä liikkuvien agenttien siirtäminen alustojen välillä (jos alusta tukee liikkuvia agentteja) [FIPA, 2000a]. AMS on agenttialustan hallinnollinen johtaja. Jos agenttialusta on hajautettu monelle palvelimelle, ulottuu AMS:n toimivalta niihin kaikkiin.

AMS ylläpitää indeksiä kaikista agenttialustalle rekisteröityneistä agenteista. Jokaisen agentin pitää rekisteröityä oman kotialustansa AMS:ään. Kotialusta on se agenttialusta, jolle agentti on käynnistetty ensimmäisen kerran. Kun agentti on rekisteröitynyt AMS:ään, se voi lähettää ja vastaanottaa viestejä MTS:n kautta. AMS:n tulee toteuttaa seuraavat toiminnot [FIPA, 2000a]:

1. *register*: Agenttien tulee rekisteröidä tunnisteensa AMS:n tietoon.
2. *deregister*: Agentit voivat poistaa tunnisteensa AMS:stä.
3. *modify*: Agentit voivat muuttaa rekisteröidyn tunnisteensa tietoja (paitsi nimi-osaa).
4. *search*: Agentit voivat etsiä toisten agenttien tietoja.
5. *get-description*. Agentit voivat tiedustella agenttialustan ominaisuuksia.

Agenttialustan AMS:lle on varattu oma tunniste, jonka nimiosa on muotoa *ams@hap*.

6.1.5 Palveluhakemisto

Palveluhakemisto (Directory Facilitator, DF) on agenttialustan pakollinen komponentti [FIPA, 2000]. Se tarjoaa agenteille keltaiset sivut -tyyppistä palvelua. Agenttialustalla voi olla yksi tai useampia palveluhakemistoita.

Jokainen agentti, joka haluaa julkistaa palveluitansa toisten käytettäväksi, rekisteröi tunnisteensa DF:n hakemistoon. Palvelun rekisteröiminen ei velvoita agenttia tarjoamaan palvelua kaikkille, vaan se voi myös kieltäytyä suorittamasta sitä [FIPA, 2000a]. DF ei siis voi taata palvelun saatavuutta eikä laatua, koska se ei voi kontrolloida agenttien elinkaarta eikä muuta sisäistä toimintaa. Tämä on osa agenttien autonomiaa.

DF:n tulee toteuttaa seuraavat toiminnot [FIPA, 2000a]:

1. *register*: Agentit voivat rekisteröidä palvelukuvauksensa ja tunnisteensa DF:n hakemistoon.
2. *deregister*: Agentit voivat poistaa palvelukuvauksensa ja tunnisteensa DF:n hakemistosta.
3. *modify*: Agentit voivat muuttaa palvelukuvauksiansa ja tunnistettansa.
4. *search*: Agentit voivat etsiä DF:n hakemistosta palvelukuvauksia.

Jos DF ei osaa vastata palveluhakuun, se voi lähettää hakupyynnön toiselle DF:lle, mikäli haun tehnyt agentti sallii sen [FIPA,2000a]. Agenttialustan oletusarvoiselle DF:lle varattu nimi on muotoa *df@hap*.

6.1.6 Kommunikaatio

FIPA-agentit keskustelevat keskenään lähettämällä toisilleen FIPA ACL -viestejä, jotka olen esitellyt aikaisemmin. Tämän lisäksi FIPA määrittelee joukon interaktioprotokollia ja yhden virallisen sisältökielen.

FIPA:n interaktioprotokollien tavoitteena on tarjota testattuja interaktiomalleja käytettäväksi erilaisiin agenttisovelluksiin sekä edistää standardoitujen protokollien uudelleenkäyttöä [FIPA, 2000c]. FIPA-agenttien ei ole pakko käyttää FIPA:n interaktioprotokollia ollakseen FIPA ACL -yhteensopivia. Mutta jos niitä käytetään, on agenttien käyttäydyttävä protokollan määrittelemällä tavalla. FIPA määrittelee yhteensä yhdeksän interaktioprotokollaa.

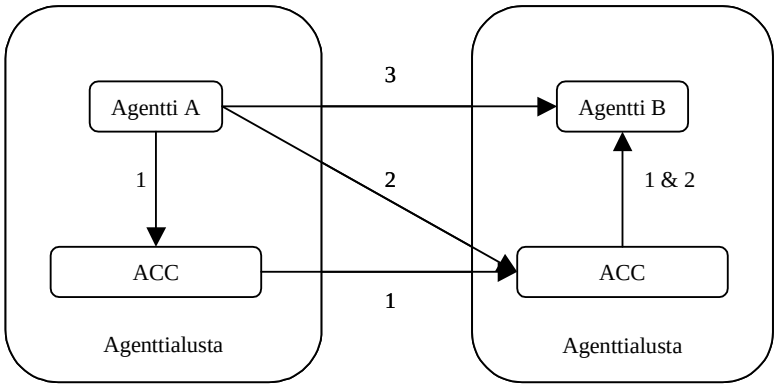
6.1.7 Viestinvälityspalvelu

Viestinvälityspalvelu (message transport service, MTS) välittää FIPA-agenttien viestejä. Jokaisella agenttialustalla on yksi viestinvälityspalvelu. Sen tarjoaa agenttikommunikaatiokanava (agent communication channel, ACC).

Viestinvälitys sisältää kolme tasoa [FIPA, 2000c]:

1. Viestinsiirtoprotokolla (message transport protocol, MTP) huolehtii viestien fyysisestä siirtämisestä kahden ACC:n välillä.
2. Viestinvälityspalvelu (MTS) kuljettaa ACL-viestejä agenttialustan agenttien välillä sekä eri agenttialustojen agenttien välillä.
3. ACL edustaa hyötykuormaa, jonka viestinvälityspalvelu ja viestinsiirtoprotokolla kuljettavat.

MTS:n välittämät viestit koostuvat kirjekuoresta sekä hyötykuormasta, joka on varsinainen FIPA ACL -viesti. Kirjekuori sisältää ainakin lähettäjän, vastaanottajan, päivämäärän ja viestin ACL-esitysmuodon. Muita parametreja voi lisätä tarpeen mukaan.



Kuva 6.3. Viestinlähetysmahdollisuudet agenttien välillä [FIPA, 2000c].

Agentilla on kolme vaihtoehtoa lähettäessään viestin toisen agenttialustan agentille, kuten kuvasta 6.3 nähdään:

1. Agentti A lähettää ensin viestin paikalliselle ACC:lle FIPA:n viestirajapinnan välityksellä tai jollain muulla tavalla. Tämän jälkeen ACC lähettää viestin agentti B:n ACC:lle, joka välittää viestin agentti B:lle.

2. Agentti A lähettää viestin suoraan agentti B:n ACC:lle, joka välittää viestin agentti B:lle. Tämä edellyttää, että agentti A on rekisteröitynyt molemmille agenttialustoille.
3. Agentti A lähettää viestin suoraan agentti B:lle. Tämä tapa ei kuulu FIPA:n määrittelemään viestinvälitykseen, vaan agenttien on itse huolehdittava viestien puskuroinnista ja mahdollisista virhetilanteista.

Kun agentti lähettää viestin ACC:n kautta, sen ei tarvitse rakentaa kirjekuorta viestin ympärille. ACC rakentaa kirjekuoren ja asettaa agentin ACL-viestin sen hyötykuormaksi. Jokainen viestinvälitykseen osallistuva ACC lisää oman leimansa kirjekuoreen. Viestit välitetään agenteille kirjekuorineen. Näin niillä on tarkka tieto siitä, mitä kautta viestit ovat tulleet. Jos viestinvälityksessä tapahtuu jokin virhe, ACC palauttaa viestin lähettäjälle virheilmoituksella varustettuna. Virheviestissä on sama keskustelutunniste, kuin alkuperäisessä viestissä.

6.1.8 AgentCities

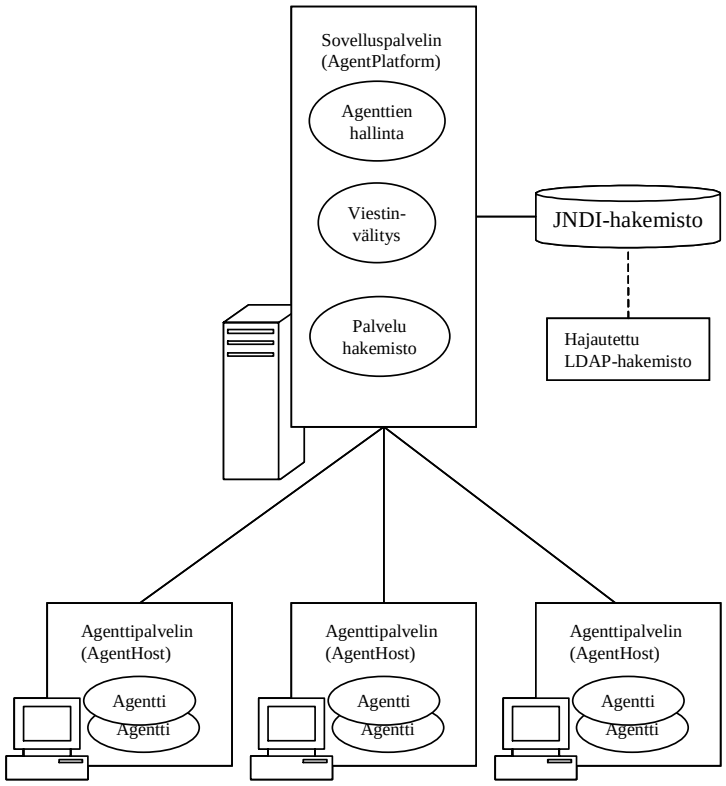
Jotta FIPA-spesifikaatioita päästäisiin kokeilemaan käytännössä, FIPA on perustanut hankkeen nimeltä AgentCities. Siinä on tarkoitus rakentaa eri puolilla maailmaa olevien FIPA-agenttialustojen verkosto. Ideana on, että kaupungeissa olisi FIPA-yhteensopivia agenttijärjestelmiä, joissa agentit tarjoavat erilaisia palveluita, kuten sähköisiä kauppapalveluita ja liiketoimintaprosessien integrointia.

Osana AgentCities -hanketta on toteutettu WWW-pohjainen palvelu, jonka avulla voi testata oman agenttialustan toiminnallisuutta. Palvelu lähettää agenttialustalle FIPA-spesifikaatioiden mukaisia palvelupyyntöjä, kuten agentin ja palvelun kuvausten rekisteröinti, muokkaaminen ja poisto. Testipalvelu lähettää sekä oikeita, että virheellisiä palvelupyyntöjä, jolloin erilaiset tilanteet tulevat huomioiduiksi. Palvelun webbisivulla on näkyvästä listasta voi tarkkailla, kuinka kauan palveluun rekisteröidyt agenttialusta ovat pysyneet pystyssä.

6.2 FIPA-arkkitehtuurin toteutus

Osittain itse suunnittelemani AgentDock -agenttiympäristö on Java-ohjelmointikielellä toteutettu FIPA-yhteensopiva agenttijärjestelmä (kuva 6.4). Järjestelmä koostuu FIPA:n määrittelemistä pakollisista agenttialustan palveluista. AMS ja DF toimivat Enterprise Java Bean (EJB) -komponentteina

sovelluspalvelimella. AgentHost-agenttipalvelin on säiliöohjelma järjestelmää käyttävien agenttisovellusten agenteille. AMS ja DF tallettavat tietonsa LDAP-palvelimelle (Light Weight Directory Access Protocol). Tätä palvelinta käytetään JNDI-rajapinnan (Java Naming and Directory Interface) välityksellä.



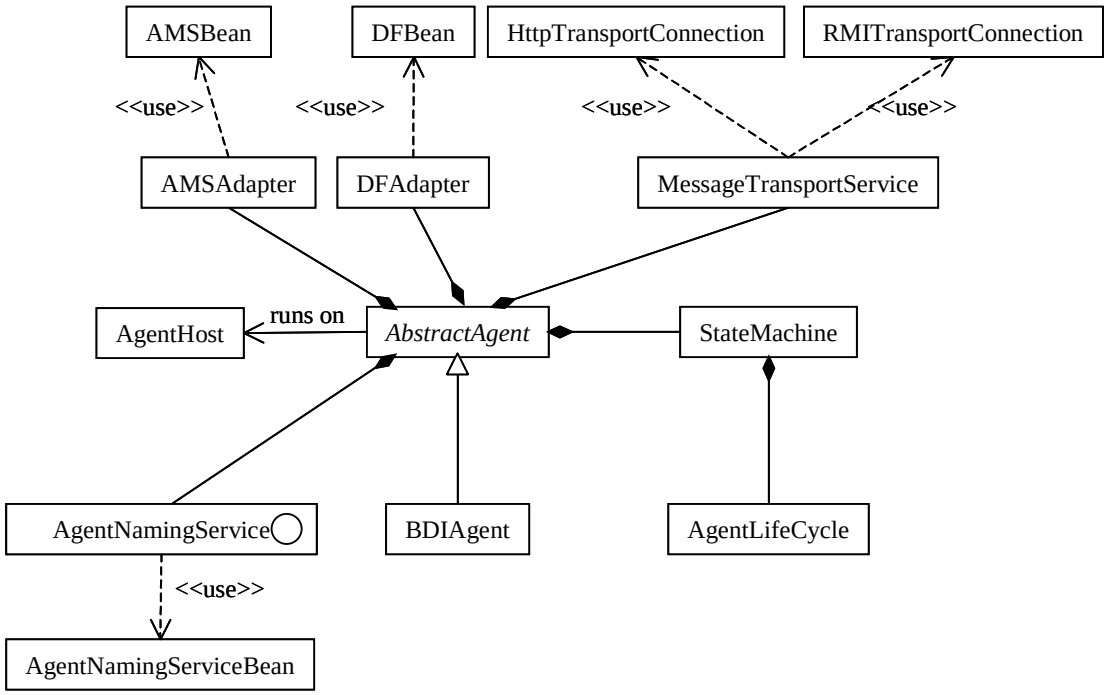
Kuva 6.4. AgentDock -järjestelmän komponentit.

6.2.1 Keskeiset luokat

Kuvan 6.5 luokkakaaviossa näkyvä AbstractAgent -luokka on isäluokka kaikille järjestelmän agenteille. Se sisältää kääreluokat, joiden kautta agentit voivat käyttää agentinhallintajärjestelmän, palveluhakemiston sekä viestinvälityspalvelun tarjoamia palveluita. Järjestelmään toteutettavat agentit perivät tämän luokan ja toteuttavat abstraktit setup- ja perform-metodit. Setup-metodia kutsutaan, kun agentti käynnistetään agenttipalvelimelle. Perform-metodi suorittaa toteuttavan agentin toiminnan. StateMachine on tilakone, joka kuvaa agentin elinkaaren agenttialustalla AgentLifeCycle-tilakarttaa käyttäen.

AgentHost on säiliöohjelma järjestelmän agenteille. Kukin agentti toimii omassa säikeessään sen sisällä. AgentHost-luokan kautta agenttikehittäjät

voivat luoda, käynnistää ja sammuttaa agentteja. Yksi AgentHost -agenttipalvelin agentteineen muodostaa yhden domainin sovellusjärjestelmässä. MessageTransportService-luokan kautta agentti voi lähettää viestejä toisille agenteille. Se käyttää sekä RMI-pohjaista, että HTTP-pohjaista viestinvälitystä, joita vastaavat luokat ovat HTTPTransportConnection ja RMITransportConnection.



Kuva 6.5. Agenttialustan keskeiset luokat.

AMSAdapter on kääreluokka AMSBeanille, joka toimii sovelluspalvelimella. Se pitää huolen, että agentin paikallinen ja AMS:llä oleva tila on sama. Lisäksi se tarjoaa agentille pääsyn AMS:n palveluihin metodikutsuina. DFAdapter toimii samoin kuin AMSAdapter. Sen kautta agentti pääsee rekisteröimään ja hakemaan palvelukuvauksia DF:ltä. AgentNamingService-luokan kautta varataan agentin ja agenttidomainin yksilölliset nimet.

6.2.2 Agentin käynnistäminen

AgenHost-agenttipalvelin on säiliöohjelma järjestelmän agenteille. Kukin agentti toimii sillä omassa säikeessään. Agentit käynnistetään palvelimelle

erityisen kehittäjäpalvelimen (DeveloperServer) avulla. Agentista rakennetaan etukäteen XML-kuvaus. Siinä kerrotaan agentin paikallinen nimi, luokan nimi, käytettävä viestinvälitysmekanismi (RMI tai HTTP) sekä agentin käyttäytymisen kuvaavan XML-tiedoston nimi. Tämä kuvaus annetaan parametrina agentin luonnin yhteydessä. Agenttipalvelin huolehtii kuvauksessa olevien parametrien alustuksesta. Kun agentti on luotu, se pitää rekisteröidä AMS:n agenttihakemistoon. Tämän jälkeen se voidaan käynnistää.

6.2.3 Agentinhallintajärjestelmä ja palveluhakemisto

Agentinhallintajärjestelmä toteuttaa FIPA AMS-spesifikaatiossa luetellut AMS:n toiminnot. Agentinhallintajärjestelmä toimii EJB-komponenttina sovelluspalvelimella, joten agenttialustan agenttien ei tarvitse lähettää sille viestejä, vaan kommunikaatio voidaan suorittaa metodikutsuilla.

Palveluhakemisto toteuttaa FIPA:n AMS-spesifikaatiossa luetellut DF:n toiminnot. Myös se toimii EJB-komponenttina sovelluspalvelimella. Järjestelmän agentit käyttävät AMS ja DF -palveluja sovitinluokan välityksellä. Tämä sovitinluokka näkyy agentille lokaalina ja se hoitaa kaikki etäkutsut agentin puolesta. Sekä AMS, että DF tallettavat tietonsa LDAP-palvelimelle.

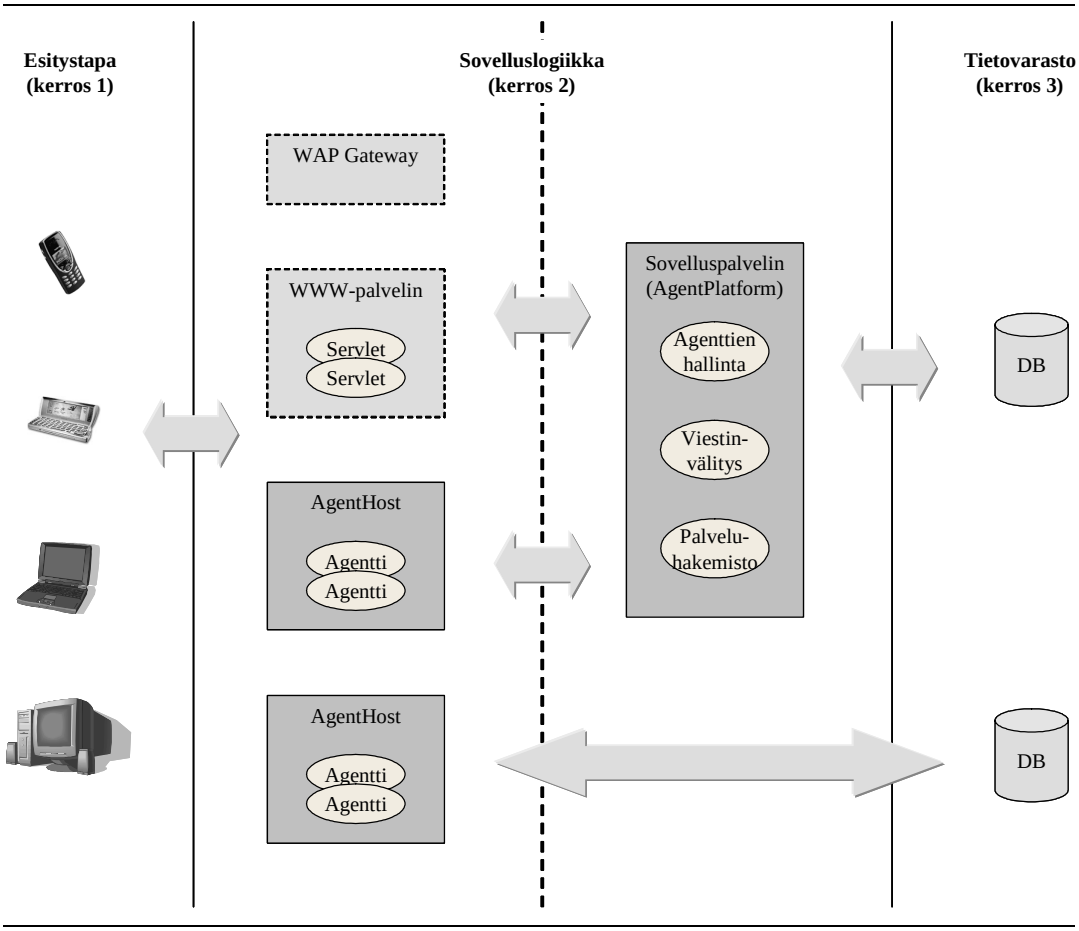
6.2.4 Viestinvälitys

Agentit voivat lähettää viestejä toisilleen viestinvälityspalvelun kautta. Jos viesti on menossa samalla palvelimella olevalle agentille, se asetetaan kyseisen agentin viestijonoon suoraan. Jos viesti on menossa toiselle palvelimelle, välitetään se kyseisen palvelimen viestipalvelulle RMI-kutsuna. Vastaanottava palvelin tallettaa viestin oikean agentin viestijonoon. Jos viesti on menossa toiselle agenttialustalle, se lähetetään HTTP-protokollalla vastaanottavan agentin agenttialustan viestijärjestelmälle, joka toimittaa sen RMI:tä käyttäen vastaanottavan agentin agenttipalvelimelle ja lopulta agentin viestijonoon.

Järjestelmän viestikielenä käytetään FIPA ACL:n XML-esitysmuotoa. Agentit käyttävät viestejä Java-olioina. Niiden ei tarvitse prosessoida XML-muotoisia viestejä, vaan tämä on viestijärjestelmän vastuulla. Agenttien pitää ainoastaan osata tulkita viestien sisältöosa. Viestijärjestelmä ei tutki viestien sisältöä, ainoastaan viestin vastaanottajan, lähettäjän ja käytettävän siirtomekanismin (RMI, HTTP). Agentit voivat lähettää viestejä toisilleen suoraan, mutta järjestelmä ei tue sitä, joten agenttikehittäjän täytyy toteuttaa se itse esimerkiksi RMI- tai sokettiratkaisuna.

6.2.5 *AgentDock-sovellusten arkkitehtuuri*

AgentDock-agenttiympäristöön toteutettava järjestelmä on kolmikerroksinen. Kuten kuvasta 6.6 havaitaan, esitystapakerros sisältää erilaiset esitystavat eri näyttölaitteille. Sovelluslogiikkakerros sisältää järjestelmään tuotettavat agentit ja erilaiset palvelinohjelmistot, kuten WWW-palvelin, sovelluspalvelin, jolla toimivat agenttialustan AMS, DF ja MTS, ja AgentHost-agenttipalvelimet agentteineen. Tietovarastokerros sisältää järjestelmän tietokannat.



Kuva 6.6. AgentDock-ympäristöön toteutettu järjestelmä.

AgentDock sopii hyvin käytettäväksi J2EE-ympäristössä (Java 2 Platform Enterprise Edition). AMS ja DF voidaan käynnistää mille tahansa sovelluspalvelimelle, joten järjestelmän integroiminen J2EE:tä käyttävän yrityksen tietojärjestelmään onnistuu melko mutkattomasti. Mitään muutoksia ei tarvita esimerkiksi tietovarastotasolla. J2EE-ympäristössä AgentDockin agentit ovat verrattavissa viestipohjaisiin EJB-komponentteihin (message driven beans, MDB) siinä mielessä, että molemmat kommunikoivat viesteillä.

6.3 Yhteenveto

FIPA määrittelee moniagenttijärjestelmässä tärkeitä palveluita, kuten agentti- ja palveluhakemistot sekä viestinvälityspalvelun. Lisäksi FIPA määrittelee agenttienvälisen viestikielen, FIPA ACL:n, sekä joukon interaktioprotokollia, joiden avulla agentit voivat käydä määrämuotoisia keskusteluja tätä kieltä käyttäen.

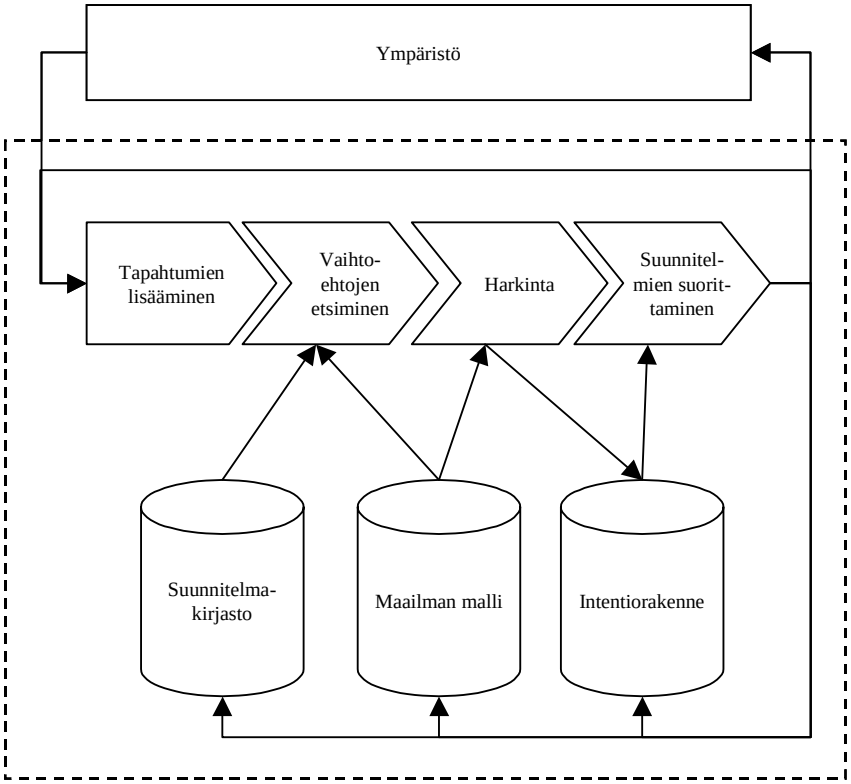
FIPA-järjestelmän palvelut voidaan toteuttaa esimerkiksi EJB-komponentteina, jolloin agenttialustan sisällä voidaan viestit välittää metodikutsuilla Java-olioina.

7 Esimerkki BDI-agentin toteutuksesta

Tässä luvussa esittelen toteuttamani luokkakirjaston, jonka avulla voi ohjelmoida BDI-agentteja. Kirjasto ei sisällä ajettavaa agenttia, vaan se on tarkoitettu uusien agenttiluokkien toteuttamiseen periyttämällä sen keskeinen luokka, BDIAgent. BDIAgent on pragmaattinen BDI-agentti, eli se ei ole suora toteutus BDI-teoriasta. BDIAgent-luokka toimii osana EJB-pohjaista FIPA-yhteensopivaa agenttijärjestelmää, joka esiteltiin luvussa 6.

7.1 Arkkitehtuurin esittely

BDI-agentti sisältää kuvassa 7.1 katkoviivan sisäpuolelle jäävät komponentit ja prosessit. Keskeiset tietorakenteet ovat suunnitelmakirjasto, maailman malli ja intentiorakenne. Agentti sisältää myös tapahtumajonon, joka ei näy kuvassa, vaan se sisältyy ”tapahtumien lisäys” -aktiviteettiin.

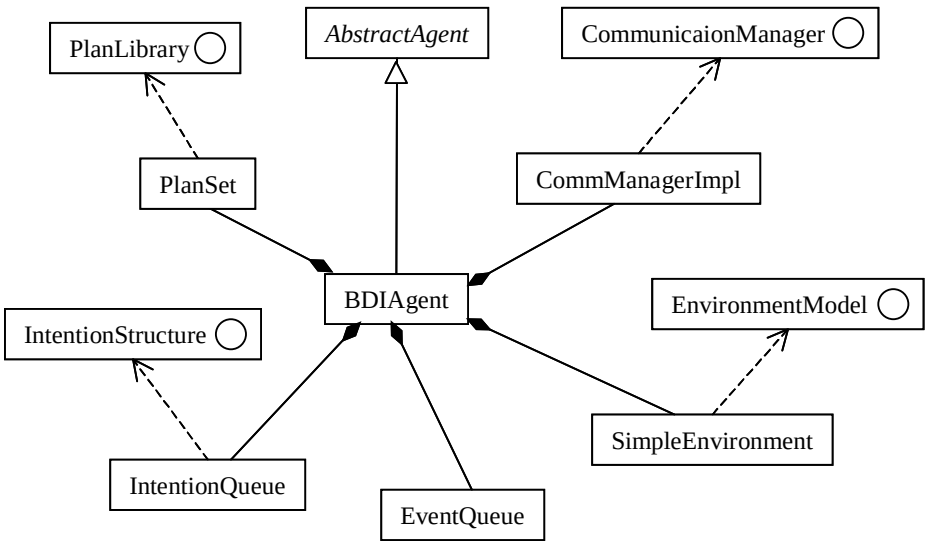


Kuva 7.1. BDI-agentin arkkitehtuuri.

Agentti säilyttää uskomuksiaan maailman mallissa. Näiden uskomusten ja tapahtumajonoon lisättyjen tapahtumien perusteella agentti tuottaa joukon vaihtoehtoisia suunnitelmia. Nämä suunnitelmat ovat agentin toiveita. Ne voivat olla keskenään ristiriitaisia, mutta kaikki agentin uskomuksiin ja tapahtumiin nähden mahdollisia. Agentti harkitsee, mikä näistä suunnitelmista on paras ja sitoutuu sen suoritukseen. Suunnitelmat voivat kohdistua joko agentin ympäristöön tai agentin sisäisiin tietorakenteisiin.

7.2 Keskeiset luokat

Kuvan 7.1 luokkakaaviossa on esitetty BDIAgent-luokka ja siihen läheisesti liittyvät muut luokat. Tapahtumajono (EventQueue) sisältää tapahtumia, joihin tulee reagoida. Koko agentin toiminta rakentuu tapahtumajonon ympärille. Agentti ylläpitää uskomuksiaan ympäristön mallissa (EnvironmentModel). Suunnitelmakirjasto (PlanLibrary) sisältää kokoelman suunnitelmia (Plan), jotka ovat alitavoitteista koostuvia reseptinomaisia listoja. Niillä agentti voi vaikuttaa ympäristöön ja itseensä. Kun agentti sitoutuu suorittamaan jonkin suunnitelman, siitä muodostetaan intentio (Intention), joka asetetaan intentiorakenteeseen (IntentionStructure) odottamaan suoritusta. Agentti voi tallentaa käymänsä keskustelut kommunikaatiomanageriin (CommunicationManager).



Kuva 6.1. BDI-toteutuksen pääluokat.

7.2.1 BDIAgent

BDIAgent-luokka perii AbstractAgent-luokan, kuten tekevät kaikki AgentDock -ympäristön agentit. Se toteuttaa AbstractAgentin abstraktit setup- ja perform-metodit. Vaikka BDIAgent on konkreettinen luokka, se on tarkoitettu perittäväksi. Perivän agentin tulee uudelleenmääritellä setup-metodi, koska etukäteen ei ole tiedossa, minkälaisia alustustoimia erilaisissa agenttisovelluksissa tarvitaan.

7.2.2 EnvironmentModel ja SimpleEnvironment

EnvironmentModel-rajapinta edustaa agentin ympäristöä. Se sisältää metodit uskomusten lisäämiseen ja poistamiseen. Lisäksi sen kautta testataan uskomusten mahdollista todistettavuutta agentin päätöksentekoprosessissa. SimpleEnvironment-luokka toteuttaa EnvironmentModel-rajapinnan yksinkertaisena listana.

7.2.3 PlanLibrary ja PlanSet

PlanLibrary-rajapinta määrittelee agentin suunnitelmakirjaston. Se sisältää metodit suunnitelmien lisäämiseen, poistamiseen sekä hakemiseen. Kirjaston suunnitelmien voi hakea joko kaikki tai tietyn tavoitteen saavuttavat. PlanSet toteuttaa PlanLibrary-rajapinnan joukkona. PlanSet sisältää PlanLibrary -rajapinnan metodien lisäksi metodit suunnitelmien lataamiseen ajoaikana, jolloin agenttia ei tarvitse sammuttaa välillä, vaan riittää, että sillä on suunnitelman luokkatiedosto saatavissa.

7.2.4 Plan, ListPlan ja Step

Plan- `ja` Step-rajapinnat ovat agentin toiminnan kannalta erittäin tärkeitä, koska niiden avulla agentti suorittaa kaikki toimintonsa. Plan-rajapinta määrittelee agentin suunnitelman. ListPlan toteuttaa Plan-rajapinnan listana tavoitteita, jotka agentti yrittää saavuttaa.

Agentin suorittamat toiminnot ovat Step-rajapinnan toteutuksia, eli askeleita. Askeleet saavuttavat aina jonkin tavoitteen. Niiden toiminta voi kohdistua agentin ympäristöön tai agenttiin itseensä.

7.3 Agentin toiminta

BDIAgent-luokka toteuttaa abstraktin perform-metodin AbstractAgent-luokasta. AbstractAgent kutsuu metodia kerran kontrollisilmukkansa aikana. BDIAgentin perform-metodin runko on mukailtu Raon ja Georgeffin [1995] esittämästä abstraktista arkkitehtuurista.

Toiminta etenee seuraavasti. Ensin selvitetään agentin vaihtoehdot. Tämä tehdään käymällä läpi tapahtumajonon tapahtumat ja vertaamalla niitä agentin käytettävissä oleviin suunnitelmiin. Jos tapahtumajonon tapahtuma sopii suunnitelman saavuttamaan tavoitteeseen ja jos suunnitelman esiehdot täyttyvät, lisätään suunnitelma vaihtoehtoihin.

Seuraavaksi tutkitaan, mikä tai mitkä vaihtoehdoista valitaan suoritukseen. Jos vaihtoehtoja on löydetty vain yksi, valitaan se suoraviivaisesti. Jos taas vaihtoehtoja on useampia, joudutaan turvautumaan niinsanottuihin metatason suunnitelmiin, joilla ristiriitatilanteessa suoritetaan valinta kilpailevien suunnitelmien välillä. Periaatteena on, että niiden johtopäätösosan tiedot kootaan listaan ja etsitään vaihtoehtoja uudestaan tämä lista syötteenä, kunnes saadaan ainoastaan yksi vaihtoehto. Jos näin ei saada ratkaistua ristiriitaa, valitaan kilpailevista vaihtoehdoista listan ensimmäinen.

Kun valittu parhaana pidetty vaihtoehtoinen suunnitelma on saatu selville, sitoudutaan sen suorittamiseen ja lisätään se intentiorakenteeseen. Jos suunnitelman tavoite on uusi, luodaan uusi intentio. Jos taas kyseessä on suunnitelma olemassa olevalle alitavoitteelle, asetetaan suunnitelma suunnitelma sen intention suunnitelmapinon päällimmäiseksi, jonka alitavoitteen uusi suunnitelma saavuttaa. Kun tämä on tehty, valitaan intentiorakenteen ensimmäisestä valittavissa olevasta intentiosta uusi alitavoite. Jos agentilla on askel, jolla tämä tavoite saavutetaan, se suoritetaan. Muutoin lisätään uusi alitavoite tapahtumajonoon. Tämän jälkeen haetaan saapuneet viestit agentin viestilistasta agenttipalvelimelta.

Lopuksi tarkistetaan agentin intentioiden tilanne. Epäonnistuneet ja saavutetut intentiot poistetaan ja päivitetään uskomukset vastaamaan tilannetta.

7.4 Uskomukset ja toiveet

Agentin uskomukset ja toiveet kuvataan tällä hetkellä merkkijonoina, joihin on liitetty totuusarvo. Uskomus ”hana-auki true” tulkitaan intuitiivisesti, että hana on päällä ja tavoite ”hana-auki true” tulkitaan, että hana halutaan päälle.

Olen suunnitellut agentin siten, että sen tietämysosaa voi laajentaa. Tulevaisuudessa on mahdollista, että agentin uskomukset pidetään esimerkiksi PROLOG-tietämyskannassa, jolloin merkkijonot sisältävät PROLOG-faktoja, kuten ”auki(hana)”. Tällöin totuusarvo jätetään pois.

7.5 Suunnitelmat

Agentin toiminnallisuus saadaan aikaan suunnitelmilla ja askelilla. Suunnitelmat ovat reseptin kaltaisia listoja, joita seuraamalla agentti saavuttaa tavoitteensa. Plan-rajapinnassa on saantimetodit sen lopullisen tavoitteen ja

ajankohtaisen alitavoitteen saamiseksi. Suunnitelma voidaan asettaa odottamaan jotain tapahtumaa, jolloin agentti odottaa, kunnes tapahtuma havaitaan tapahtumajonossa ja jatkaa suoritusta vasta sen jälkeen. Tällöin koko suunnitelman sisältävä intentio asetetaan odottamaan. Tämä on kätevää esimerkiksi kun on lähetetty viesti toiselle agentille ja odotetaan siihen vastausta. Agentti voi suorittaa toisia intentioita tällä välin.

Suunnitelma sisältää tavoitteen, esiehdot, vaikutuksen sekä joukon alitavoitteita. Suunnitelman tavoite toimii laukaisimena suunnitelman valinnalle. Lisäksi esiehtojen tulee täyttyä. Kun suunnitelma on suoritettu, lisätään tai poistetaan vaikutusosan uskomukset.

Step, eli askel, saavuttaa yhden alitavoitteen. Agentin askeleet on talletettu avain/arvo-pareina, joissa avaimina on Goal-luokan ilmentymiä ja arvoina Step-toteutusluokkien ilmentymiä. Jos suunnitelman alitavoitteelle ei ole sen saavuttavaa askelta, lisätään tapahtumajonoon uusi tavoite ja etsitään myöhemmin uudella kontrollisilmukan kierroksella suunnitelma, jolla se saavutettaisiin. Askeleen toiminnan kohde voi olla agentin ympäristö tai agentin sisäiset tietorakenteet ja suunnitelmat. Askelten toteutuksessa on otettava huomioon seuraavat rajoitukset:

1. Toteutus ei saa sisältää pitkäkestoisia silmukoita, koska agentti ei yksisäikeisenä pysty tekemään tällä välin mitään muuta. Näin ollen agentille olennainen reaktiivisuus kärsii pahasti. Tämä hankaloittaa jonkin verran askelten suunnittelua ja ohjelmointia.
2. Toteutuksen täytyy palauttaa tieto suorituksen onnistumisesta, jotta agentti tietää mitä tehdä suorituksen jälkeen. Askel voidaan suorittaa uudestaan, jos ensimmäinen kerta epäonnistuu.

Kun agentti sitoutuu suorittamaan suunnitelman, se sitoutuu sekä tavoitteeseen että keinoihin, eli suunnitelman runkoon. Jos valittu suunnitelma epäonnistuu, tavoite asetetaan uudestaan tapahtumajonoon ja etsitään uusi suunnitelma. Tavoitteeseen lisätään tieto siitä, mikä suunnitelma on jo kokeillut sen saavuttamista. Tämä otetaan huomioon vaihtoehtoja kartoitettaessa.

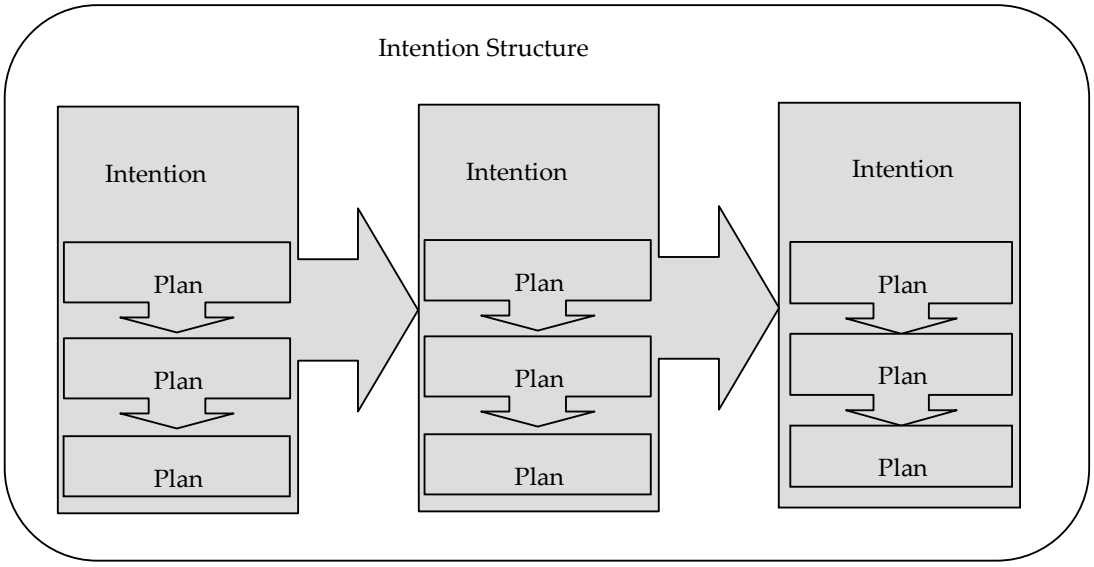
7.6 Intentiot

BDIAgentin intentiot on toteutettu Intention-luokan ilmentyminä. Intention-luokka on rakennettu siten, että se sisältää pinon, johon asetetaan suunnitelmia. Pinon alimmaisena on intention luonnin yhteydessä annettu alustava suunnitelma.

Kuvassa 7.3 näkyy tapa, jolla intentiot on organisoitu. Kun intentiorakenteeseen lisätään uusi suunnitelma, tutkitaan suunnitelman

tavoitetta. Jos se on jonkin intention alitavoite, suunnitelma asetetaan kyseisen intention suunnitelmapinon päällimmäiseksi. Muutoin luodaan uusi intentio ja asetetaan se jonoon viimeiseksi.

Seuraava suoritettava intentio valitaan jonon alusta. Jos intentio on odotustilassa, koska sen päällimmäinen suunnitelma odottaa jotakin tapahtumaa, valitaan jonossa seuraavana oleva intentio. Kun intention päällimmäinen suunnitelma on suoritettu kokonaan, se poistetaan pinosta ja jatketaan sen alapuolella olevan suunnitelman suorittamista siitä, mihin jäätiin edellisen alitavoitteen asettamisen jälkeen. Suunnitelmien abstraktiotaso kasvaa siis pinoa alaspäin mentäessä. Pinon pohjalla saattaa olla vain summittainen suunnitelma, joka sisältää pelkästään uusia suunnitelmia generoivia alitavoitteita.



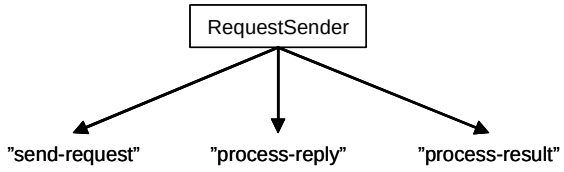
Kuva 7.3. Intentiorakenteen organisaatio.

7.7 Esimerkki interaktioprotokollan suorittamisesta

Interaktioprotokollat, kuten muutkin BDI-agentin toiminta, toteutetaan suunnitelmilla. Yksinkertaistettuna esimerkkinä käytän FIPA:n [2001] määrittelemää request-protokollaa, joka etenee seuraavasti:

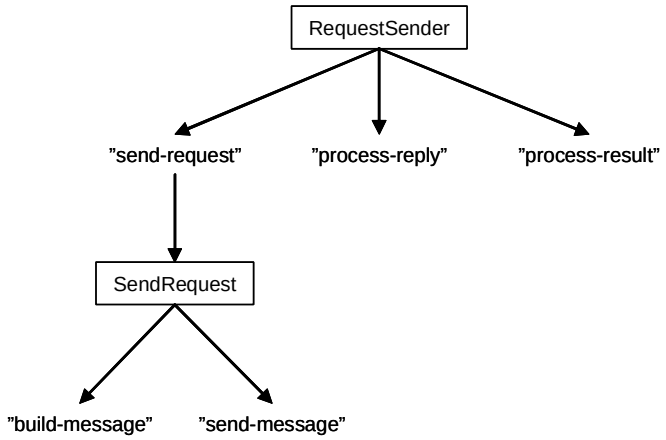
1. Agentti a vaatii agentti b:tä suorittamaan jonkin tehtävän lähettämällä tälle ACL-viestin, jonka kommunikatiivisena aktina on request ja sisältönä lauseke, joka kuvaa vaadittavan toiminnon.
2. Jos agentti b ei ymmärrä viestin sisältöä, se vastaa “not understood”. Jos se jostain syystä kieltäytyy suorittamasta tehtävän, se vastaa ”refuse”. Jos se suostuu suorittamaan tehtävän, se vastaa ”agree”.
3. Kun b on suorittanut tehtävän, se vastaa agentille ”inform-done” -viestillä, jos se ei palauta tulosta tai ”inform-ref”-viestillä, jos se palauttaa jonkin tuloksen. Jos b epäonnistuu, se lähettää ”failure”-viestin.

Interaktioprotokollasta täytyy toteuttaa sekä aloitteen tekevä, että vaatimuksen vastaanottava puoli. Tässä esimerkissä kuvaan vain lähettävän puolen suunnitelmat. Kuvassa 7.5 näkyy aloitteen tekevän osapuolen osittainen suunnitelma ”RequestSender”, joka sisältää kolme alitavoitetta: ”send-request”, ”process-reply” ja ”process-result”.



Kuva 7.5. FIPA Request -protokollan lähettävä osapuoli.

Kun suunnitelmaa suoritetaan, tarkastellaan ensin ”send-request”-tavoitetta. Oletetaan, että agentilla ei ole askelta, jolla se saavuttaisi tavoitteen suoraan, joten se lisätään agentin tapahtumajonoon. Onnistuneen päättelyn tuloksena agentti löytää suunnitelmakirjastostaan SendRequest -suunnitelman, joka lisätään intentiorakenteeseen ja päästään tilanteeseen, joka näkyy kuvassa 7.6. SendRequest sisältää tavoitteet ”build-message” ja ”send-message”. Jotta “send-request”-tavoite täyttyisi, on näiden molempien täytyttävä.

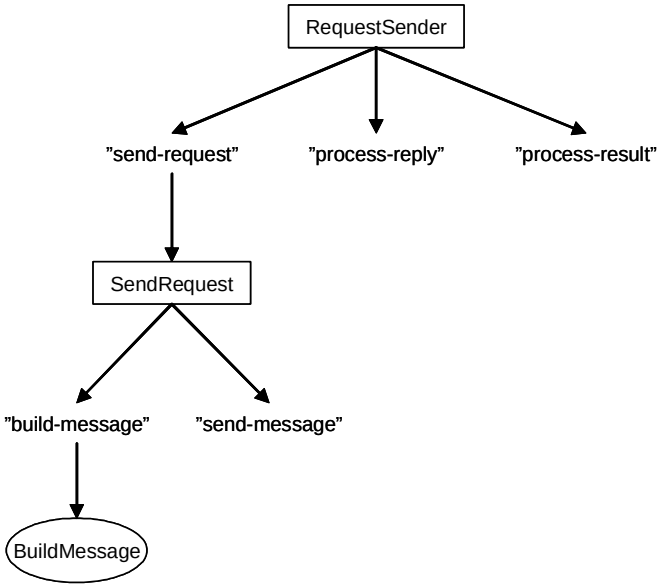


Kuva 7.6.

Seuraavaksi tarkastellaan tavoitetta ”build-message”. Tässä esimerkissä agentilla on BuildMessage -askel, jonka suorittamalla agentti saavuttaa ”build-message”-tavoitteen. Tämä tilanne näkyy kuvassa 7.7. BuildMessage-askel yksinkertaisesti rakentaa uuden ACL-viestin.

Kun BuildMessage-askel on suoritettu, tarkastellaan ”send-message”-tavoitetta. Oletetaan, että agentilla on SendMessage-askel, joka saavuttaa sen. Tämä askel lähettää viestin. Koska RequestSender-suunnitelmaa ei voida jatkaa, ennen kuin request-viestin vastaanottava agentti vastaa, asettaa askel viimeisenä toimenaan RequestSender-suunnitelman odottamaan ”message-arrived”-tapahtumaa.

Kun ”message-arrived”-tapahtuma havaitaan tapahtumajonossa, jatketaan RequestSender -suunnitelman suorittamista. Seuraavaksi tarkasteluun otetaan ”process-reply”-tavoite. Taaskin oletamme, että agentilla on ProcessReply-askel, joka saavuttaa tavoitteen ja tulkitsee saapuneen viestin. Jos viestin performatiivi on ”refuse” tai ”not-understood”, voi askel tulkita tilanteen virheeksi ja palauttaa virheilmoituksen, joka johtaa RequestSender -suunnitelman epäonnistumiseen. Toinen vaihtoehto on asettaa ”send-request” -tavoite uudestaan tapahtumajonoon, jolloin agentti yrittää suorittaa SendRequest -suunnitelman uudestaan. Jos taas performatiivi on ”agree”, ProcessReply -askel asettaa suunnitelma odottamaan ”result-arrived”-tapahtumaa.

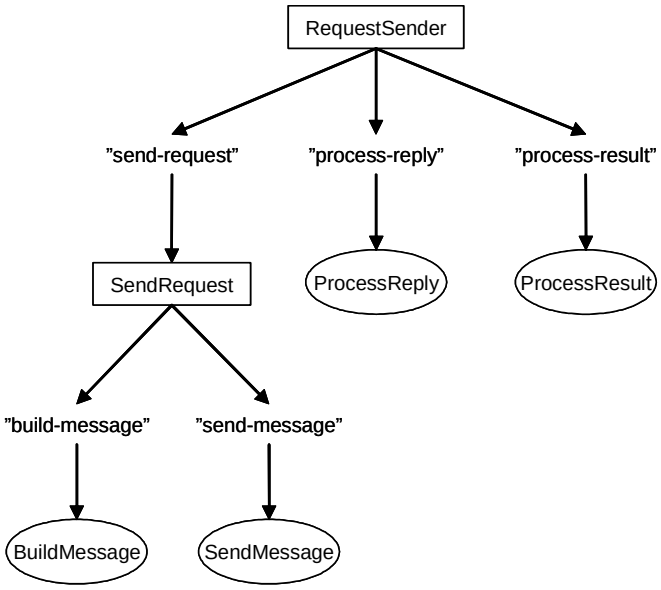


Kuva 7.7.

Kun tapahtuma havaitaan, jatketaan SendRequest-suunnitelman suorittamista. Taas oletetaan, että agentilla on ProcessResult-niminen askel, joka käsittelee saapuneen vastauksen mahdollisesti generoiden uusia tavoitteita tai uskomuksia. Tämän jälkeen suunnitelma on suoritettu ja se poistetaan intentiorakenteesta.

Kuvassa 7.8 näkyy tarkentunut suunnitelma kokonaisuudessaan. Kuvasta havaitaan, että suunnitelmista ja alisuunnitelmista on muodostunut suorituksen edetessä puumainen rakenne, jota käydään suorituksessa läpi syvyysjärjestyksessä. Lapsisolmuja evaluoidaan, kunnes löytyy lehtisolmu, joka voidaan suorittaa heti.

Seuraavalla kerralla suoritettuna suunnitelma voi rakentua eri lailla riippuen siitä, onko sillä vaihtoehtoinen suunnitelma SendRequest-suunnitelmalle. Tällöin agentti voi päätyä päättelyssään siihen. Askelten suhteen ei päde sama. Yhden tavoitteen saavuttaa täsmälleen yksi askel. Jos siis halutaan vaihtoehtoisia toimintoja, ne pitää toteuttaa suunnitelmina.



Kuva 7.8. RequestSender -suunnitelman lopputilanne.

7.8 Käyttöliittymän lisääminen

BDIAgent ei sisällä käyttöliittymää. Sellainen voidaan toteuttaa perivään luokkaan. Jos halutaan tehdä normaali Java-käyttöliittymä, se pitää käynnistää joko setup-metodissa tai jossakin askeleessa omaan säikeeseensä. Käyttöliittymälle annetaan kahva agentin tapahtumajonoon, johon se voi kirjoittaa käyttäjän komennoista generoituja tavoitteita.

Jos agentille halutaan tehdä WWW-käyttöliittymä, pitää käyttäjän komennot lähettää ensin esimerkiksi servletin käsiteltäväksi. Tämä servlet sitten muuntaa komennot agentille ACL-viestinä lähetettäviksi tavoitteiksi. Agentilla tulee tietenkin molemmissa tapauksissa olla sopivat suunnitelmat tavoitteiden saavuttamiseksi.

7.9 Konfigurointi

BDIAgentin konfiguroinnissa voidaan käyttää hyväksi AgentSetupTool-työkaluluokkaa, joka lukee XML-tiedostosta agentin käyttämien askeleiden ja suunnitelmien kuvaukset.

```
<agent name="Agentti" agent-class="DrinkAgent">
   <step-definitions>
      <step name="drink" value="true"
         class="com.minutor.agent.bdiagent.test.TestStep"/>
      <step name="open-tap" value="true"
         class="com.minutor.agent.bdiagent.test.TestStep"/>
   </step-definitions>
   <plan name="drink-water" class-name="com.minutor.agent.bdiagent.plan.ListPlan">
      <pre value="have-glass" value="true"/>
      <pre value="have-soda" value="false"/>
      <goal value="quenched-thirst" truth="true"/>
      <effect value="quenched-thirst" operation="add" truth="true"/>
      <steps>
         <step name="open-tap"/>
         <step name="drink"/>
      </steps>
    </plan>
</agent>
```

Kuva 7.9. Osa agentin konfiguraatiotiedostoa.

Kuvassa 7.9 näkyvässä XML-kuvauksessa kerrotaan agentin käytössä olevat Step-rajapinnan toteuttavien luokkien nimet ja niiden saavuttamat tavoitteet sekä suunnitelmien kuvaukset. Suunnitelmaosassa kerrotaan kunkin suunnitelman tavoite, alitavoitteet, esiehdot ja vaikutus.

7.10 Toteutuksen arviointi

BDIAgent ja sen apuluokat toteuttavat eräänlaisen pragmaattisen BDI-agentin. Se ei siis toteuta BDI-teoriaa täydellisesti. Esimerkiksi intentioiden roolit käytännön järkeilyssä eivät kaikki täyty.

BDI-agentin arkkitehtuurissa olen ottanut vaikutteita PRS-järjestelmästä, joka on muodostunut jonkinlaiseksi referenssitoteutukseksi BDI-agentista. Suurin ero siihen ja muihin tutkimiini BDI-arkkitehtuureihin nähden on, että omassa toteutuksessani ei käytetä tulkattavaa kieltä, jolla suunnitelmat esitetään, vaan kaikki on toteutettu Javalla. Myös metatason suunnitelmat toimivat eri periaatteella PRS:ään verrattuna.

Testauksissa tuli ilmi joitakin ongelmia agentin toiminnassa. Agentti ei esimerkiksi aina huomaa, jos jokin sen suunnitelmista jumiutuu esimerkiksi yllä esitetyssä interaktioprotokollassa. Agentti tunnistaa kyllä virhetilanteet viestien lähettämisessä ja muissa omissa toiminnoissa, mutta jos se odottaa vastausta toiselta agentilta ja tämä toinen agentti kaatuu, jää vastausta odottava intentio jumiin. Tämä ei haittaa agentin muiden toimintojen suorittamista, mutta vie muistiresursseja.

Kuten interaktioesimerkistä nähdään, interaktioprotokollien toteuttaminen intentiona ja suunnitelmina voi olla monimutkaista. Tämä todetaan myös FIPA:n [2000c] spesifikaatioissa. Interaktioprotokollien tulisi perustua jonkinlaiseen tilakoneeseen, kuten Pitt ja Mamdani [1999] ehdottavat. Jotta

interaktioprotokollat ja muut agentin keskenään vuorovaikutuksessa olevat suunnitelmat tulisvat suunniteltua järkevästi, olisi syytä kehittää jonkinlainen mallinnusmenetelmä. Tällaisia mallinnusmenetelmiä on kehitetty UML-pohjalta ja agenttijärjestelmäkohtaisina. Uskon, että jokin UML:n stereotyyppejä luovasti käyttävä kuvaustyyli sopisi käytettäväksi tässä yhteydessä.

Agentiella ei ole toteutuksessani tällä hetkellä ollenkaan ajan tajua. Tämä aiheuttaa ongelmia intentioiden toteuttamiskelpoisuuden ja validiuden tarkastelussa. Jonkinlainen aikaleima auttaisi mm. edellämainitussa intention jumiutumistilanteessa. Agentin tavoitteet ovat saavutustavoitteita. Lisäksi tarvittaisiin ainakin ylläpito- ja testaustavoitteet. Ylläpitotavoite tarkoittaa, että pyritään pitämään jokin arvo tietyllä määritellyllä alueella tai pyritään pitämään jonkin proposition arvo totena. Testaustavoite tarkoittaa yksinkertaisisesti, että pitääkö jokin propositio paikkansa. Näillä kahdella tavoitteella saataisiin toteutettua ilmaisuvoimaisempia suunnitelmia.

Arkkitehtuuri mahdollistaa ns. top down -toteutuksen, jolloin voidaan tehdä ensin alustavia suunnitelmia ja kokeilla järjestelmää niillä. Tarvittaessa voidaan sitten lisätä korjattuja, tarkempia suunnitelmia ilman, että aiempia pitää poistaa. Arkkitehtuuri sopii kohtalaisen hyvin erilaisten prototyyppien toteuttamiseen. Se vaatii tosin koko AgentDock -järjestelmän asentamista, mikä pitää sisällään sovelluspalvelimen ja LDAP-palvelimen konfiguroimisen ja käynnistämisen. Tästä syystä järjestelmää ei ehkä kannata käyttää kovin pienten prototyyppien toteutukseen.

Toinen sovelluskohde voisi olla suunnitteluprosessin tukeminen. Prosessiin osallistuville ihmisille kullekin oma agentti, joka edustaisi omistajaansa prosessissa. Järjestelmässä voisi olla yksi isäntäagentti, joka toimisi järjestelmän ylimpänä auktoriteettina jakaen tehtäviä muille agenteille ja niiden kautta ihmisille. Ihmisiä edustavat agentit voisivat neuvotella keskenään suoritettavien aktiviteettien jakamisesta, jos näyttää siltä, että joku ei ehdi suorittaa omaa tehtäväänsä ajoissa.

AgentDock-järjestelmällä olisi mielenkiintoista toteuttaa AgentCities -hankkeeseen palveluita. Tällöin tulisi kuitenkin toteuttaa ensin esimerkiksi PrologBasedBDIAgent, joka perisi BDIAgent-luokan ja sisältäisi agentin uskomukset Prolog-faktoina.

8 Lopuksi

Vaikka agenttipohjaisessa ohjelmistokehityksessä on havaittavissa pieniä hiipumisen merkkejä 1990-luvun loppuun nähden, on se silti erittäin mielenkiintoinen tapa tuottaa ohjelmistoja. Kun agentit ajatellaan intentionaalisina komponentteina, saadaan uusi abstraktiotaso ohjelmistojen toiminnan kuvaamiseen. Voidaan puhua järjestelmän toimijoiden uskomuksista ja aikomuksista. Uskon, että tämä helpottaa monimutkaisten järjestelmien suunnittelua erityisesti silloin, kun sitä on suunnittelemassa eri alojen edustajia.

Termille agentti ei ole löytynyt yhteistä kaikkien hyväksymää määritelmää. On vain joitakin ajatuksia siitä, mitkä ominaisuudet kuvaavat hyvin agentteja. Lähdekirjallisuuden perusteella agentteja on toteutettu moniin eri teorioihin ja arkkitehtuureihin perustuen, mikä tarkoittaa, että ne voidaan myös määritellä monien eri teorioiden kautta. Luvussa kolme esittelin joitakin lähestymistapoja rationaalisen toiminnan tuottamiseen. Näistä lähestymistavoista BDI-teoria tarjoaa selkeän ja mielenkiintoisen pohjan agenttien toteuttamiseen.

Jotta agenttiparadigma tulisi yleisemmin käyttöön ohjelmistotuotannossa, tarvitaan standardeja agenttienväliseen viestintään. FIPA on tällä hetkellä ainoa agenttistandardeja tuottava yhteisö. Se on tuottanut spesifikaatioita agenttien ja palveluiden hallintaan sekä agenttienväliseen viestintään. Esittelin tutkielmassani näitä spesifikaatioita ja niiden mukaan toimivan agenttijärjestelmän. Lisäksi esittelin kehittämäni BDI-agentin arkkitehtuurin, joka on suunniteltu toimivaksi FIPA-ympäristössä. BDI-malli on käyttökelpoinen FIPA-agenttien toteutuksessa, koska FIPA:n määrittelemien performatiivien semantiikka kuvataan BDI-käsitteiden avulla.

Tutkielman tuloksena kehittämääni BDI-arkkitehtuuria ei ole testattu monimutkaisten järjestelmien toteutuksessa ja sitä voidaan pitää melko kokeellisena. Se on kuitenkin mielestäni käyttökelpoinen esimerkiksi prototyyppien tekemiseen. Tulevaisuudessa arkkitehtuuria voisi kehittää esimerkiksi lisäämällä siihen kunnon tietämyskannan, jolloin uskomusten esittäminen ja niistä johtopäätösten tekeminen saisi enemmän ilmaisuvoimaa. Toinen kehityssuunta voisi olla käyttöliittymän kehittely. Luvussa seitsemän esitin kaksi tapaa, jolla sellainen voitaisiin toteuttaa.

Viiteluettelo

[Aikio ja Vornanen, 1992] Annukka Aikio (toim.) uusinut Rauni Vornanen, *Uusi sivistyssanakirja.* Otava, 1992.

[Bratman, 1990] Michael E. Bratman, What is Intention? In: Cohen, Morgan, Pollack (ed.), *Intentions in Communication*, The MIT Press: Cambridge, MA, 1990.

[Bratman et al., 1988] Michael E. Bratman, David J. Israel and Martha E. Pollack, Plans and Resource-Bounded Practical Reasoning. *Computational Intelligence*, **4**, 4 (1988), 349-365.

[Brickley and Guha, 2003] Dan Brickley and R.V. Guha (eds.), RDF Vocabulary Description Language 1.0: RDF Schema, *W3C Working Draft* (2003), http://www.w3c.org/TR/rdf-schema/.

[Brooks, 1985] Rodney A. Brooks, A robust layered control system for a mobile robot, *AI Memo 864*, MIT Artificial Intelligence Laboratory (1985).

[Brooks, 1991] Rodney A. Brooks, Intelligence without representation. *Artificial Intelligence* **47**, 1-3 (1991), 139-151.

[Bussman and Muller, 1992] S. Bussman and J. Muller, A Negotiation Framework for Co-operating Agents. In: S. M. Deen (ed.), *Proc. CKBS-SIG*, Dake Centre, University of Keele (1992), 1-17.

[Cohen and Levesque, 1990] Philip R. Cohen and Hector J. Levesque, Intention is Choice with Commitment, *Artificial Intelligence* **42** (1990), Elsevier Science Publishers, 213-261.

[Durfee, 1989] E. H. Durfee, *Coordination of Distributed Problem Solvers*. Kluwer Academic Publishers: Boston, MA, 1989.

[Durfee, 1999] Edmund H. Durfee, Distributed Problem Solving and Planning. In: Gerhard Weiss (ed.), *Multiagent systems: a modern approach to distributed artificial intelligence*. The MIT Press: Cambridge, MA, 1999.

[Farquhar et al., 1996] A. Farquhar, R. Fikes, and J. Rice. The Ontolingua server: A tool for collaborative ontology construction. Technical report, Stanford KSL 96-26, 1996.

[Ferguson, 1992] Innes A. Ferguson, *TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents*. Technical Report No. 273, University of Cambridge Computer Laboratory, 1992.

[Ferguson, 1994] Innes A. Ferguson, Integrated control and coordinated behavior: a case for agent models. In: *Intelligent Agents: proc. of ECAI-94 Workshop on Agent Theories, Architectures, and Languages, Lecture Notes in Artificial Intelligence* **890** (1994), Springer-Verlag, 203-218.

[Fikes and Nilsson, 1971] R. E. Fikes and N. J. Nilsson, STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, **2,** 3-4 (1971),189-208.

[FIPA, 2000a] FIPA Agent Management Specification, Foundation for Intelligent Physical Agents (2000), http://www.fipa.org/specs/fipa00023.

[FIPA, 2000b] FIPA Communicative Act Library Specification, Foundation for Intelligent Physical Agents (2000), http://www.fipa.org/specs/fipa00037.

[FIPA 2000c] FIPA Interaction Protocol Library Specification, Foundation for Intelligent Physical Agents (2000), http://www.fipa.org/specs/fipa00025.

[FIPA, 2001] FIPA Request Interaction Protocol Spesification, Foundation for Intelligent Physical Agents (2001), http://www.fipa.org/specs/fipa00026.

[FIPA, 2002] ACL Message Structure Specification, Foundation for Intelligent Physical Agents (2992), http://www.fipa.org/specs/00061.

[Fischer et al, 1995] Klaus Fischer, Jörg P. Müller and Markus Pischel, A Pragmatic BDI architecture. In: *Intelligent Agents II: agent theories, architectures, and languages, proceedings of IJCAI '95 Workshop (ATAL),* Montréal, Canada, August 1995*; Lecture Notes in Artificial Intelligence* **1037** (1995), Springer-Verlag, 203-218.

[Franklin and Graesser, 1996] Stan Franklin and Art Graesser, Is it an agent, or just a program?: a taxonomy for autonomous agents. In: *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*, Springer-Verlag, 1996.

[Georgeff and Ingrand, 1989] Michael P. Georgeff and Fracois Félix Ingrand, Decision-Making in an Embedded Reasoning System. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI'89),* Detroit, MI, August 1989.

[Georgeff and Lansky, 1987] M. P. Georgeff and A. L. Lansky, Reactive reasoning and planning. *Proceedings of the Sixth National Conference on Artificial Intelligence* (AAAI-87) (1987), 677-682.

[Gero and Stanton, 1988] John S. Gero and R. Stanton (eds.), *Artificial Intelligence Developments and Applications*. Elsevier Science Publishers B.V., 1988.

[Green et al., 1997] Shaw Green, Leon Hurst, Brenda Nangle, Pádraig Cunningham, Fergal Somers and Richard Evans. Software Agents: A review. Saatavana osoitteesta http://citeseer.nj.nec.com/green97software.html.

[Guarino, 1997] Nicola Guarino, Semantic Matching: Formal Ontological Distinctions for Information Organization, Extraction, and Integration. In: Maria Teresa Pazienza (ed.), *Information Extraction: A Multidisciplinary*

Approach to an Emerging Information Technology, *International Summer School, SCIE-97*, Frascati, Italy (1997), *Lecture Notes in Computer Science* **1299** (1997), Springer, 139-170.

[Horvitz et al., 1988] Eric J. Horvitz, John S. Breese and Max Henrion, Decision Theory in Expert Systems and Artificial Intelligence. In: *International Journal of Approximate Reasoning* (1988), 2:247-302.

[Huber, 1999] Marcus J. Huber, JAM: A BDI-theoretic mobile agent architecture. In: *Proceedings of the Third International Conference on Autonomous Agents (Agents '99)* (1999), 236 - 243.

[Jennings, 1993] Nicholas R. Jennings, Commitments and Conventions: The Foundation of Coordination in Multi-Agent Systems. *The Knowledge Engineering review* **8**, 3 (1993), 223-250.

[Jennings et al., 1998] Nicholas R. Jennings, Katia Sycara and Michael Wooldridge, A roadmap of agent research and development. *Autonomous agents and multi-agent systems* **1** (1998), 275-306.

[Jennings, 2000] Nicholas R. Jennings, On agent-based software engineering. *Artificial intelligence* **117** (2000), Elsevier, 277-296.

[Labrou et al., 1999] Yannis Labrou, Tim Finin and Yun Peng, Agent Communication Languages: The Current Landscape. *IEEE Intelligent Systems*, (March/April 1999), 45-52.

[Labrou and Finin, 1997] Yannis Labrou and Tim Finin, Semantics for an Agent Communication Language, In: *Intelligent Agents V: Agent Theories, Architectures, and Languages, proc. of ATAL'97, Lecture Notes in Artificial Intelligence* **1365** (1997), Springer-Verlag, 209-214.

[Lassila and Swick, 1999] Ora Lassila and Ralph R. Swick, Resource Description Framework (RDF) Model and Syntax Specification, *W3C Recommendation* (February 1999), http://www.w3c.org/TR/REC-rdf-syntax/.

[LivingSystems, 2002] Living Markets: Industrial and Commercial Applications of Agents, esitelmä FIPA-konferenssissa, Lausanne 13.2.2002.

[Müller et al., 1994] Jörg P. Müller, Markus Pischel and Michael Thiel. Modeling reactive behaviour in vertically layered agent architectures. In: *Intelligent Agents: proceedings of ECAI-94 Workshop on Agent Theories, Architectures, and Languages; Lecture Notes in Artificial Intelligence* **890** (1994), Springer-Verlag, 261-276.

[Newell and Simon, 1963] Newell and Herbert A. Simon, GPS, a program that simulates human thought. In: Edward A. Feigenbaum and Julian Feldman, *Computers and Thought* (New York: McGraw-Hill, 1963), 279 - 293.

[Noy and McGuinness, 2001] Natalya F. Noy and Deborah L. McGuinnes, Ontology Development 101: A Guide to Creating Your First Ontology. *Stanford Knowledge Systems Laboratory Technical Report KSL-01-05, Stanford Medical Informatics Technical Report SMI-2001-0880* (March 2001).

[Nwana et al., 1996] H. S. Nwana, L. Lee and N. R. Jennings, Co-ordination in software agent systems. *BT Technol J.* **14,** 4 (October 1996), 79-89.

[Odell, 1999] James Odell. Objects and agents: how do they differ? Artikkeli saatavana osoitteesta http://www.jamesodell.com/publications.

[OMG, 2000] Agent technology green paper. OMG Document agent/00-09-01 version 1.0, September 2000.

[OMG, 2003] OMG Unified Modeling Language Specification. OMG Document formal/03-03-01 version 1.5, March 2003.

[Perrault and Allen, 1980] C. Raymond Perrault and James F. Allen, A plan-based analysis of indirect speech acts. *American Journal of Computational Linguistics* **6**, 3-4 (July-December 1980).

[Pitt and Mamdani, 1999] Jeremy Pitt and Abe Mamdani, Communication Protocols in Multi-Agent Systems. *Workshop on Specifying and Implementing Conversation Policies, Autonomous Agents '99* (1999).

[Pollack, 1990] Martha E. Pollack, Plans as complex mental attitudes. In: P.R. Cohen, J. Morgan and M.E. Pollack (eds.), *Intentions in Communication*, MIT Press (1990).

[Pollack, 1992] Martha E. Pollack, The uses of plans. *Artificial Intelligence* **53**, 1 (1992), 43-68.

[Rao and Georgeff, 1995] Anand S. Rao and Michael P. Georgeff, BDI Agents: from Theory to Practice. *Technical Note 56*, AAII (April 1995), *Proceedings of the First International Conference on Multiagent Systems (ICMAS-95),* San Francisco, USA (June 1995).

[Russel and Norvig, 1995] Stuart J. Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 1995.

[Sandholm, 1999] Tuomas W. Sandholm, Distributed Rational Decision Making. Teoksessa Gerhard Weiss (ed.), *Multiagent systems: a modern approach to distributed artificial intelligence*. The MIT Press: Cambridge, MA, 1999.

[Searle, 1979] John R. Searle, *Expression and Meaning, Studies in the Theory of Speech Acts*. Cambridge University Press, 1979.

[Sycara, 1998] Katia P. Sycara, Multiagent Systems. *AI Magazine* **19**, 2 (Summer 1998), 79-92.

[Turocy and von Stengel] Theodore L. Turocy and Bernhard von Stengel, Game theory. In: *Encyclopedia of Information*, Academic Press (2002).

[Weiss (toim.), 1999] Gerhard Weiss, *Multiagent Systems, A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 1999.

[Wooldridge and Jennings, 1994] Michael Wooldridge and Nicholas R. Jennings, Agent theories, architectures, and languages: a survey, *Intelligent Agents: Proceedings of ECAI-94 Workshop on Agent Theories, Architectures, and Languages, Lecture Notes in Artificial Intelligence* **890**, Springer-Verlag (1994), 1-39.

[Wooldridge, 1997] Michael Wooldridge, Agent-based software engineering. *IEEE Transactions on software engineering* **144**, 1(February 1997), 26-37.

[Wooldridge, 1999] Michael Wooldridge, Intelligent agents. In: Gerhard Weiss (ed.), *Multiagent systems: a modern approach to distributed artificial intelligence*. The MIT Press: Cambridge, MA, 1999.

[Wooldridge, 2000] Michael Wooldridge. *Reasoning about Rational Agents*. MIT Press, 2000.

[Wooldridge, 2002] Michael Wooldridge. *Introduction to Multiagent Systems*. John Wiley & Sons, 2002.