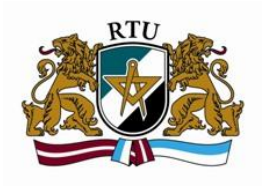


IEGULDĪJUMS TAVĀ NĀKOTNĒ

Projekts „Daudzaģentu robotizētas intelektuālas sistēmas tehnoloģijas izstrāde”
Vienošanās Nr. 2010/0258/2DP/2.1.1.1.0/10/APIA/VIAA/005
PVS ID 1528

Programmatūras apraksts – daudzaģentu programmatūra

Saturs

Ievads

Dokumenta mērķis. Dokumenta mērķis ir aprakstīt atbilstošo nodevumu.

Darbības sfēra. Dokuments paredzēts publiskai lietošanai.

Ievads problēmsfērā. Eksistē roboti, kuri katrs pats par sevi ir spējīgi izpildīt kādu noteiktu uzdevumu, piemēram, robots – putekļu sūcējs var iztīrīt noteiktu telpu (piemēram, nelielu dzīvojamo istabu) atbilstošā laikā. Tomēr, ja ir veicams kāds apjomīgs uzdevums (piemēram, noliktavas tīrīšana), kuru individuāli pieņemamā laikā viens robots izpildīt nevar, rodas nepieciešamība apvienot vairāku robotu spēkus. Robotiem, kuri radīti individuāla uzdevuma risināšanai, parasti nav iebūvētu mehānismu, ar kuru palīdzību mijiedarboties ar citiem robotiem. Lai šāda veida robotus apvienotu kopēja mērķa sasniegšanai, ir jāizstrādā vadības mehānisms, kas apvienotu atsevišķus robotus vienotā sistēmā, kā arī veiktu uzdevumu sadali starp robotiem tā, lai panāktu vēlamo rezultātu izpildot atbilstošus kritērijus.

Šajā dokumentā aprakstītais nodevums ir daudzaģentu sistēmas programmatūras modulis, kas izstrādāts ar mērķi nodrošināt daudzu robotu sistēmas vadības funkciju.

Nodaļu pārskats. Šī dokumenta pirmajā nodaļā ir aprakstīts nodevums, tā būtība, kā arī dots nodevuma pielietojuma piemērs. Otrajā nodaļā ir noslēgums, ietverot nodevuma pielietojuma aprakstu, kā arī nodevuma aprobāciju. Dokumentam pievienots izmantotās literatūras saraksts, kā arī atzinība.

1. Nodevuma izklāsts

1.1. Nodevuma būtība

Nodevuma priekšmets ir daudzaģentu sistēmas programmatūras modulis. Šis modulis sastāv no divām loģiskām daļām – daudzaģentu sistēmas un programmatūras grafiskās lietotāja saskarnes nodrošināšanai. Abas minētās daļas ir aprakstītas turpmākajās divās apakšnodaļās.

1.1.1. Daudzaģentu sistēma

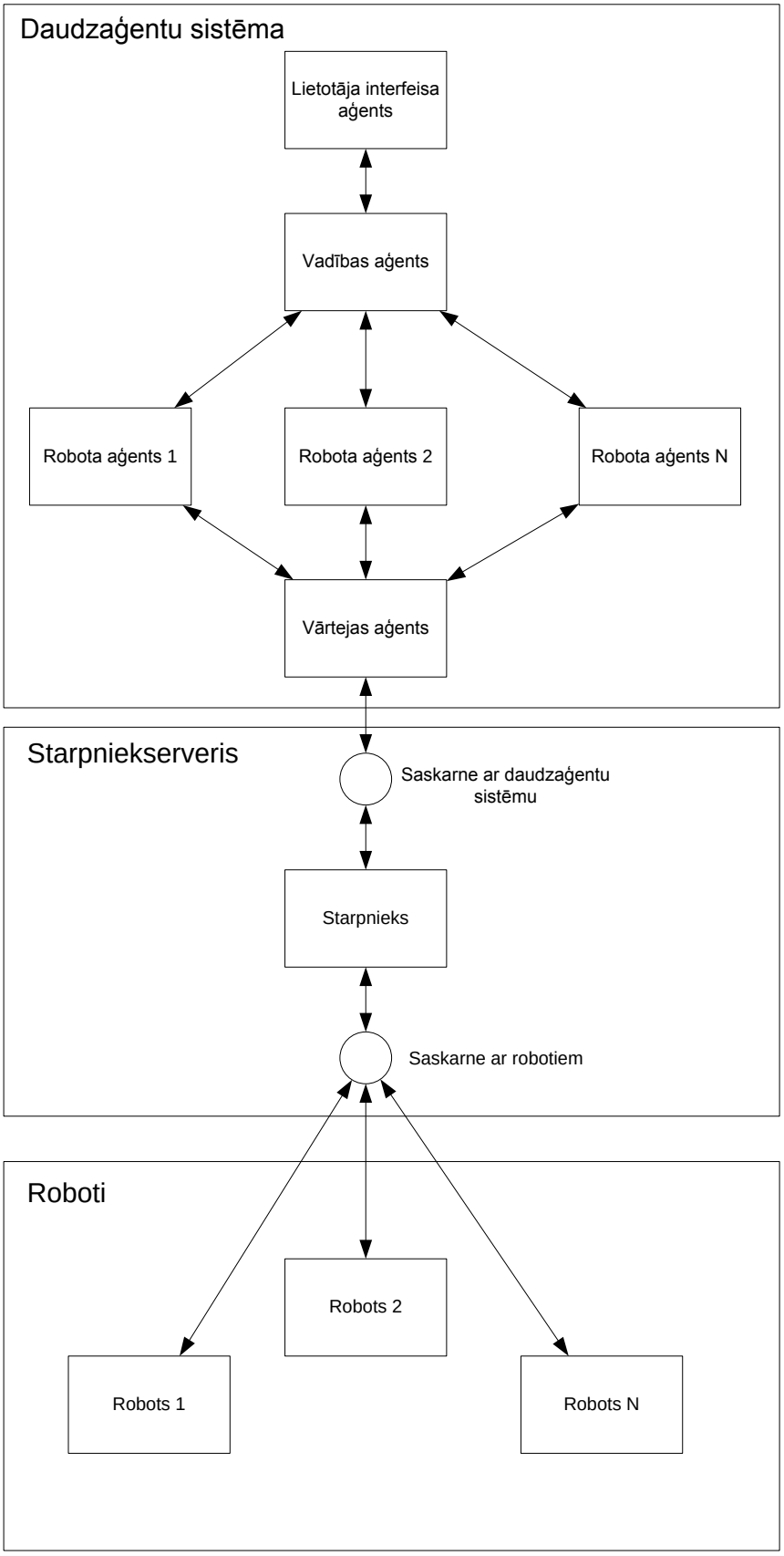
Daudzaģentu sistēma pilda daudzu robotu sistēmas vadības mehānisma funkciju. Daudzu robotu sistēmas kopējā arhitektūra ir parādīta 1.1. att.

Šajā dokumentā aprakstītais nodevums ir tieši daudzaģentu sistēmas slānis, kas sastāv no daudzaģentu sistēmas, kur katrs aģents reprezentē atbilstošo robotu fiziskajā vidē. Papildus tam daudzaģentu sistēmā ir ietverts arī lietotāja interfeisa aģents, kas nodrošina saskarni ar lietotāju, vārtejas aģents, kurš nodrošina daudzaģentu sistēmas saskarni ar starpniekserveri, kā arī vadības aģents, kas nodrošina lietotāja specificēto uzdevumu vadību.

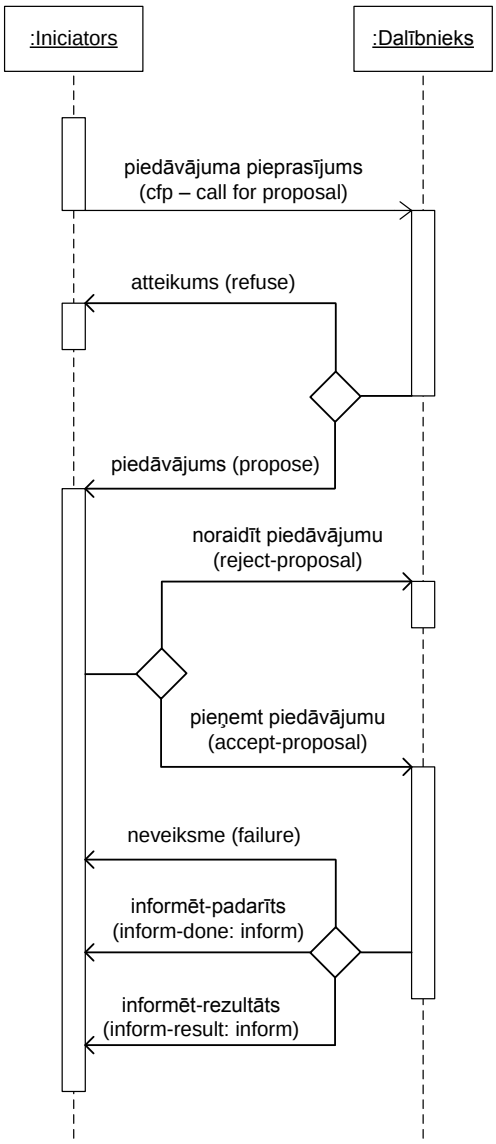
Daudzaģentu sistēma ir atbildīga par lietotāja uzdoto uzdevumu izpildi un robotu vadību augstā līmenī, kā arī pieņem ilgtermiņa lēmumus par robotu aģentu darbību. Tā operē ar robota X un Y koordinātēm un virzienu telpā, kā arī lietotāja uzdotajām apstrādājamās telpas koordinātēm, lai veiktu uzdevumu sadali un kontrolētu to izpildi.

Daudzaģentu sistēmas darbības algoritms ir šāds. Daudzaģentu sistēma no lietotāja saņem izpildāmo uzdevumu ar lietotāja interfeisa aģenta starpniecību. Lietotāja interfeisa aģents šo uzdevumu tālāk pārsūta vadības aģentam, kurš uzdevumu (ja tas nepieciešams) sadala mazākos uzdevumos (apakšuzdevumos) tā, ka katru no tiem var atsevišķi veikt viens robots. Pēc tam šie uzdevumu tiek sadalīti starp robotu aģentiem izmantojot atbilstošo uzdevumu sadales mehānismu. Kad robota aģents saņem uzdevumu, tas uzsāk šī uzdevuma izpildi. Pēc uzdevuma izpildes, robota aģents par to paziņo vadības aģentam. Kad visi atbilstošā uzdevuma apakšuzdevumi ir izpildīti, vadības aģents par uzdevuma izpildi informē lietotāja interfeisa aģentu, kas, savukārt, par to informē lietotāju. Par uzdevumu sadales mehānismu izstrādātajā programmatūrā kalpo Contract NET protokols [1] (sk. 1.2. att.).

Contract NET protokolā piedalās viens iniciators un viens vai vairāki dalībnieki. Sarunas iniciators vēlas, lai kāds no sarunas dalībniekiem izpilda noteiktu uzdevumu, un ir ar mieru uzticēt uzdevuma izpildi dalībniekam, kurš solīs iespējami mazāku maksu par dotā uzdevuma izpildi.



1.1. att. Sistēmas kopējā arhitektūra (aizgūts no [2])

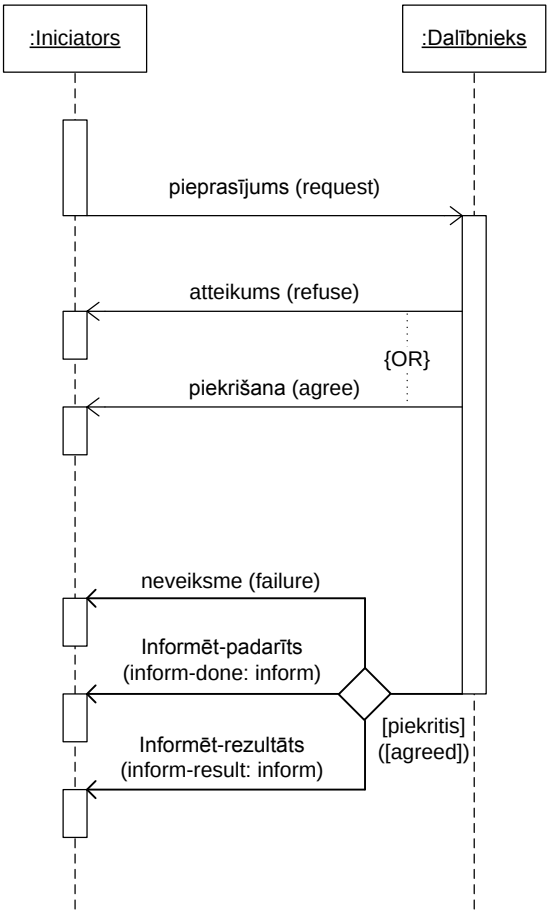


1.2. att. Contract NET protokols (aizgūts no [3])

Sarunas iniciators uzsāk protokolu nosūtot piedāvājuma pieprasījumu atbilstošajiem dalībniekiem. Katrs dalībnieks novērtē savas iespējas izpildīt norādīto uzdevumu un atbild ar piedāvājumu, piedāvājot noteiktu maksu par dotā uzdevuma izpildi. Dalībnieks var arī pamatoti atteikties izpildīt uzdevumu (piemēram, ja tā rīcībā nav atbilstošo resursu, zināšanu, u.tml.). Kad sarunas iniciators ir saņēmis piedāvājumus (vai atteikumus) no visiem sarunas dalībniekiem (vai arī tad, ja ir iestājusies noildze), tas veic saņemto piedāvājumu izvērtēšanu un nosaka labāko piedāvājumu (to, kurā solīta zemākā cena). Labākā piedāvājuma autora piedāvājums tiek pieņemts, bet pārējo dalībnieku piedāvājumi tiek noraidīti. Kad sarunas dalībnieks saņem informāciju par to, ka tā sūtītais piedāvājums ir pieņemts, tas uzsāk atbilstošā uzdevuma izpildi. Pēc uzdevuma izpildes,

dalībnieks informē sarunas iniciatoru par rezultātu vai arī neveiksmi, ja uzdevumu izpildīt nav izdevies [3]. Izstrādātajā programmatūrā sarunas iniciatora lomu spēlē vadības aģents, bet sarunas dalībnieku lomas – robotu aģenti. Šādi veidota uzdevumu sadales mehānisma mērķis ir atrast robotu, kurš ar pēc iespējas zemākām izmaksām var izpildīt atbilstošo uzdevumu.

Saziņai starp lietotāja interfeisa aģentu un vadības aģentu tiek izmantots FIPA Request protokols (sk. 1.3. att.). Šajā protokolā pieprasījuma iniciators sūta pieprasījumu (request) pieprasījuma dalībniekam. Pieprasījuma dalībnieks atbild ar atteikumu (refuse) vai piekrišanu (agree). Ja dalībnieks ir piekritis pieprasījuma izpildei, tad tas veic pieprasījumā norādīto darbību izpildi un informē iniciatoru par pieprasījuma izpildes rezultātu. Iniciatora lomu šajā gadījumā spēlē lietotāja interfeisa aģents, kurš no lietotāja saņem pieprasījumu par uzdevuma izpildi un pārsūta to tālāk sarunas dalībniekam – vadības aģentam. Vadības aģents, savukārt, organizē uzdevuma izpildi, pielietojot robotu aģentus kā izpildītājus. Par uzdevuma (sekmīgas vai nesekmīgas) izpildes rezultātu vadības aģents informē lietotāja interfeisa aģentu.



1.3. att. FIPA Request protokols (aizgūts no [3])

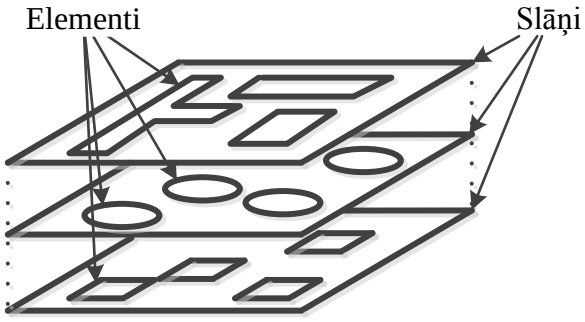
Saziņa starp robotu aģentiem un vārtejas aģentu tiek organizēta līdzīgā veidā. Šajā gadījumā atbilstošā robota aģents spēlē iniciatora lomu, pieprasot vārtejas aģentam izpildīt noteiktu komandu uz robotu aģentam piesaistītā robota. Vārtejas aģents spēlē dalībnieka lomu attiecīgi atbildot uz šiem pieprasījumiem.

Papildus jau aprakstītai komunikācijai pastāv vēl viens ziņojumu veids – robotu statusa ziņojumi. Šie ziņojumi ietver informāciju par robotu pašreizējo atrašanās vietu, pašreiz izpildāmo uzdevumu, kā arī šī uzdevuma izpildes statusu. Minēto informāciju izmanto robotu aģenti robotu vadības nodrošināšanai, kā arī lietotāja interfeisa aģents sistēmas kopainas attēlošanai lietotājam. Šos ziņojumus izsūta vārtejas aģents, kurš periodiski saņem informāciju no robotiem. Robotu statusa ziņojumi ir vienvirziena ziņojumi, un to pārsūtīšanai netiek izmantots specifisks saziņas protokols.

Daudzaģentu sistēma ir realizēta JADE [4] vidē izmantojot Java programmēšanas valodu.

1.1.2. Lietotāja saskarnes modulis

Lietotāja saskarnes pamatā ir ideja, ka jebkura karte sastāv no vairākiem slāņiem, kur katrs slānis satur kāda noteikta veida objektus (skat. 1.1. att.). Šāda uzbūve ir vispārīga un plaši pielietojama – atšķiras tikai slāņos iekļauto elementu noformējums (piemēram, forma un krāsa).



1.1. att. Kartes veidošana no slāņiem (aizgūts no [5]).

Vispārīgā arhitektūra

Kartes lietojuma izstrādei ir izvēlēts Eclipse GEF (Graphical Modelling Framework) izstrādes ietvars. Tā pamatā ir MVC (Model-View-Controller) princips, kur datu struktūras ir neatkarīgas no skata veidošanas datu struktūrām. Kontrolleris seko līdzi datu stāvoklim un izmaiņu gadījumā atjauno skatu, lai tas atbilstu izmaiņām. Šāda uzbūve ļauj efektīvi reaģēt uz izmaiņām, jo izmaiņu neesamības gadījumā lieki neveic skata atjauninašanu, taču ja izmaiņas ir notikušas, tad atjauno tikai atbilstošās skata daļas.

Projekta implementācijā ir izdalīti divi neatkarīgi moduļi – datu struktūras bibliotēka un grafiskais lietojuma interfeiss. Datu struktūras bibliotēku var iekļaut kā atkarību citu komponenšu

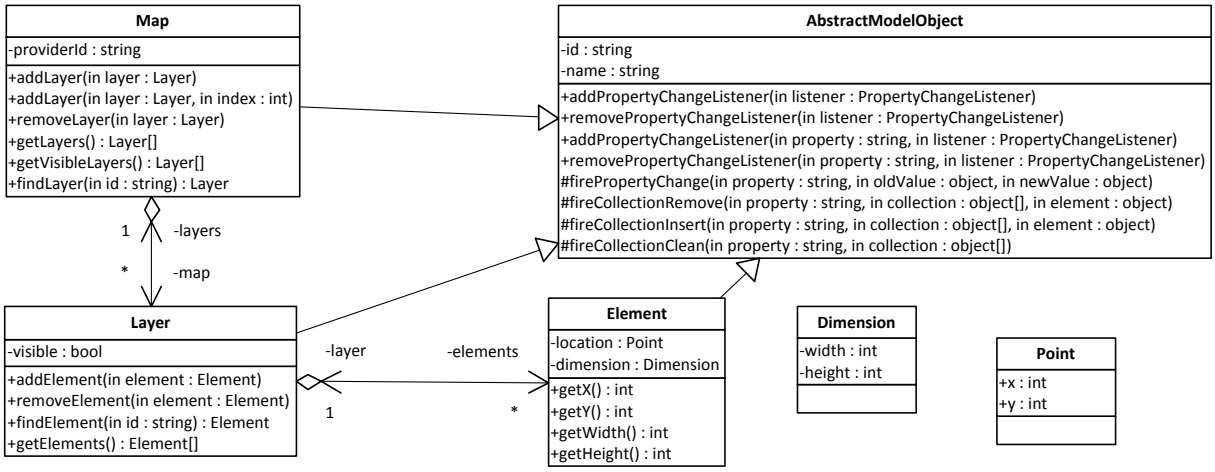
izstrādei bez nepieciešamības iekļaut arī grafiskā lietojuma bibliotēkas kā transitīvās atkarības. Grafiskais lietojuma interfeiss (GUI – Graphical User Interface) nodrošina:

- karšu saraksta un datu struktūras attēlošanu kokveida sarakstā,
- kartes attēlošanu izmantojot konkrētās kartes nodrošinātās figūras,
- elementu atribūtu attēlošanu parametru skatā.

Datu struktūra

Īpaša nozīme datu struktūrās ir *AbstractModelObject* klasei, kuru izmanto kā pamata klasi visām kartes datu struktūrā izmantotajām klasēm (skat. 1.2. att.), jo tā implementē metodes izmaiņu paziņošanai un izmaiņu klausītāju pievienošanai. Tādā veidā ir nodrošināta nepieciešamā infrastruktūra izmaiņu paziņojumu izplatīšanai – kontrolleri var reģistrēt kā izmaiņu klausītāju un datu klasei ir nodrošināti līdzekļi izmaiņu izziņošanai.

Īpaša uzmanība jāpievērš kolekcijas tipa elementu izmaiņu reģistrēšanai – standarta Java utilītklases nepiedāvā risinājumus, kā noteikt kurš elements kolekcijā ir mainīts (piemēram, no slāņa ir jāizņem vai jāpievieno elements) un ciklā pārbaudīt visu kolekciju ir neefektīvi. Šīs problēmas risināšanai izveidotas metodes *fireCollectionRemove* un *fireCollectionInsert*, kuras izplata īpašu *CollectionPropertyChangedEvent* objektu, kas satur informāciju par (1) kolekciju, kas mainīta, (2) mainīto objektu un (3) izmaiņu veidu (pievienošana vai izņemšana no kolekcijas).

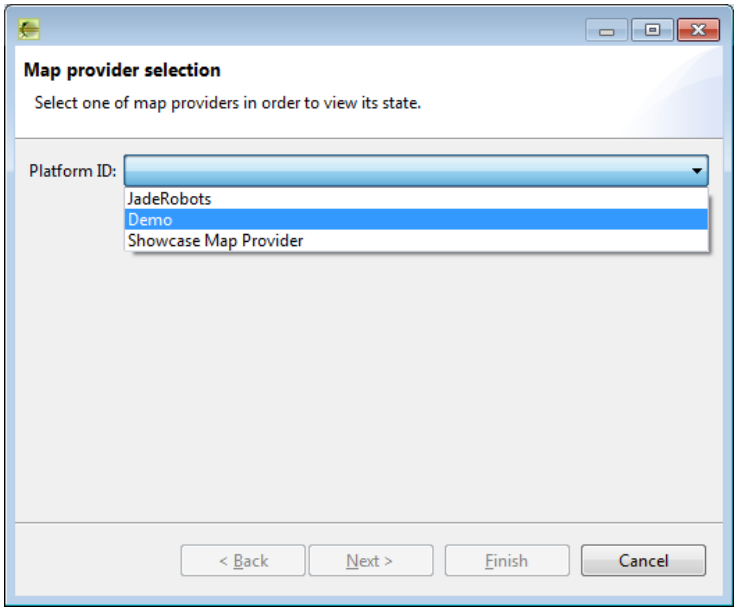


1.2. att. UML klašu diagramma (aizgūts no [5]).

Datu struktūra arī definē īpašu Java anotāciju, kuru var izmantot, lai atzīmētu tos klases elementus, kuri ir jāattēlo īpašību skatā. Anotāciju var izmantot, lai apzīmētu *getX* tipa metodes (kur *X* ir parametra vārds). Anotācijai atsevišķi var norādīt kādu konkrētu parametra nosaukumu, taču ja tāds nav norādīts, tad mēģina izgūt parametra nosaukumu no metodes nosaukuma, pieņemot, ka ir izmantots Java valodā ierastais *Camel-Case* nosaukumu veidošanas stils.

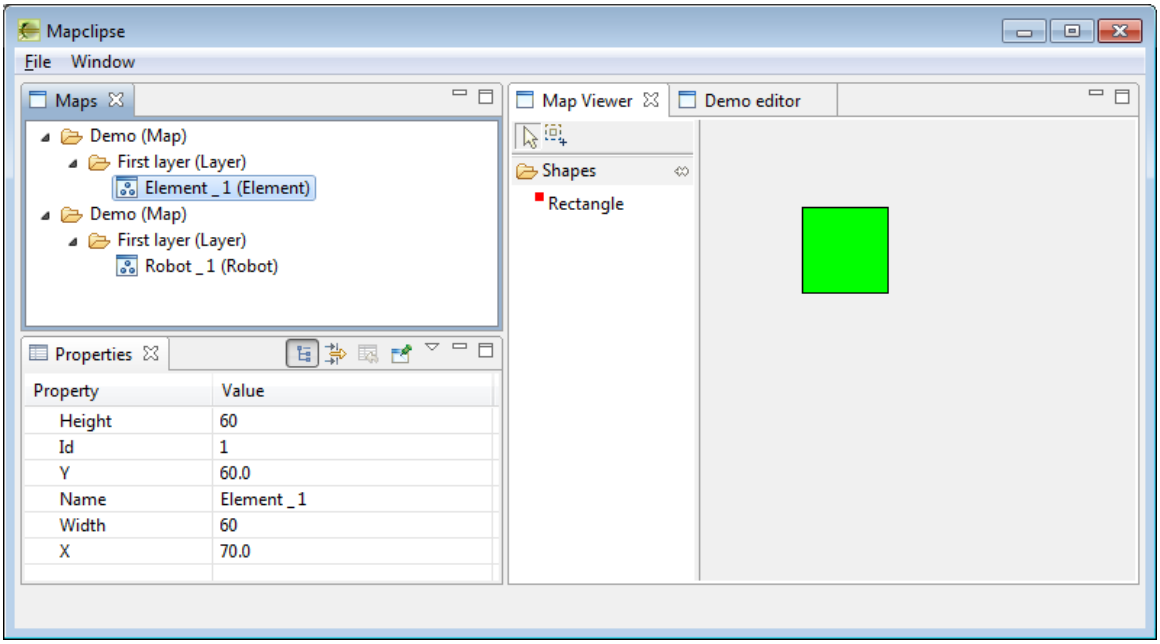
Spraudņi

Katra kartes implementācija ir kā Eclipse spraudnis, kas inicializācijas laikā reģistrē servisu, kas implementē *IEditPolicyInstaller* interfeisu (tas faktiski ir augstākā līmeņa kontrolleris). Kartes atvēršanas laikā skenē visus reģistrētos servisus (vienlaicīgi tādi var būt vairāki) un piedāvā kā opcijas kartes pievienošanas vednī (skat. 1.3. att.), kas pieejams caur [File]→[Add Map…] izvēli.



1.3. att. Kartes veida izvēle.

Atkarībā no kartes specifikas, pēc kartes veida izvēles, izvēlētais spraudnis var attēlot papildus konfigurācijas logus (piemēram, servera savienojuma parametrus). Pēc korektu vērtību norādīšanas vednī, „Maps" sadaļā pievienojas jauns augstākā līmeņa elements, kas reprezentē pievienoto karti (skat. 1.4. att.). Karšu sarakstā veicot dubultklikšķi, tiks atvērta karte redaktora logā, kur tā var tikt apskatīta un vajadzības gadījumā arī rediģēta (ja konkrētās kartes spraudnis atbalsta šādu funkcionalitāti).



1.4. att. Mapclipse ar parauga kartēm.

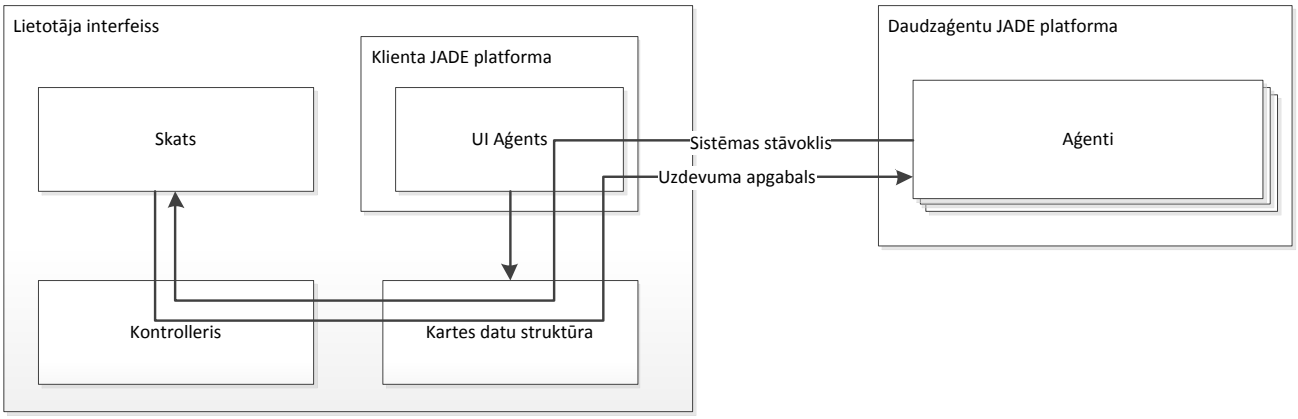
Kā redzams 1.4. attēlā, tad iezīmētā elementa (Element_1, kas ir izvēlēts „Maps" sadaļā) ir atribūti un to vērtības ir attēlotas „Properties" sadaļā. Tikai tie objektu atribūti, kuri ir atzīmēti ar *@Property* anotāciju, ir attēloti šajā sarakstā.

Intelektuālo aģentu karte

Daudzaģentu sistēmas stāvokļa attēlošanai ir izveidots Eclipse spraudnis. Kartes datu struktūra sastāv no 3 veidu elementiem (tātad, arī 3 slāņiem):

1. Robotiem piešķirto apgabalu figūras (zemākajā slānī)
2. Iezīmētā apgabala figūras (vidējā slānī)
3. Robotu figūras (augšējā slānī)

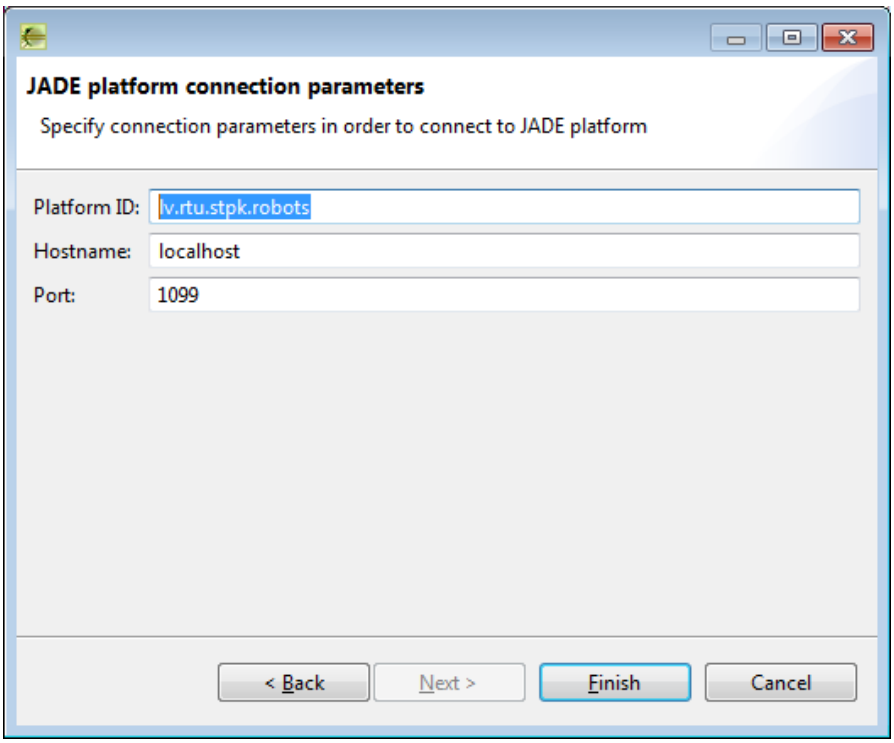
Lai izgūtu daudzaģentu sistēmas stāvokli, spraudņa inicializēšanas laikā tiek palaista JADE platforma, kas darbojas kā klienta (pakļautā) platforma daudzaģentu sistēmas platformai. Šajā klienta platformā ir izvietots īpašs UI (User Interface) aģents, kas sazinās ar daudzaģentu sistēmas aģentiem, izmantojot ACL (Agent Communication Language) protokolu, un atjauno datu struktūru balstoties uz saņemto informāciju (skat. 1.5. att.). Ja kartē ir norādīts kāds aģenta darbības apgabals, tad UI aģents to nodod vadības aģentam, kas to sadala mazākos uzdevumos katram robota aģentam.



1.5. att. Datu plūsma starp skatu un daudzaģentu sistēmu.

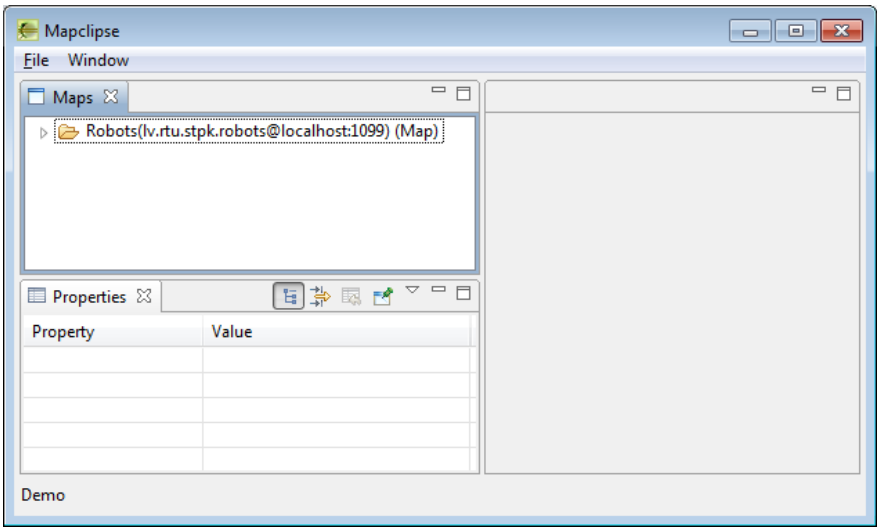
1.2. Nodevuma pielietojuma piemērs

Nodevuma pielietojuma piemērā aprakstīts, kā iegūt precīzas robota koordinātu izmaiņas laika gaitā. Lai sasniegtu mērķi, jāatver karti, ko var izdarīt galvenajā izvēlnē atverot [File] → [Add Map…], pēc kā atveras 1.3. attēlā redzamais dialoga logs. Lai pievienotu robotu sistēmas karti, vajag izvēlēties „JadeRobots" un nospiest „Next >". Pēc tam jāievada daudzaģentu sistēmas platformas savienojuma parametri. Pēc noklusējuma daudzaģentu sistēmas platforma tiek palaista ar platformas identifikatoru „lv.rtu.stpk.robots" uz 1099. porta. Ja platforma darbojas uz lokālā datora ar noklusējuma parametriem, tad parametri ir jāaizpilda ar vērtībām, kas redzamas 1.6. attēlā.

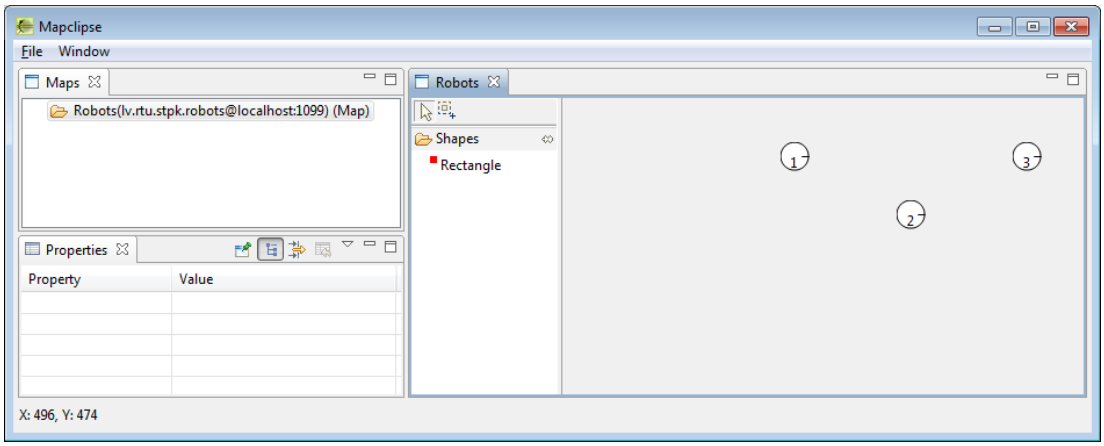


1.6. att. Daudzaģentu sistēmas platformas savienojuma parametru logs.

Veiksmīga savienojuma gadījumā tiek izveidots savienojums un „Maps” sadaļā parādās jauns elements, kā tas redzams 1.7. attēlā. Veicot dubultklikšķi uz augšējā līmeņa elementiem, galvenajā logā atveras karte ar robotu izvietojumu tajā (skat. 1.8. att).

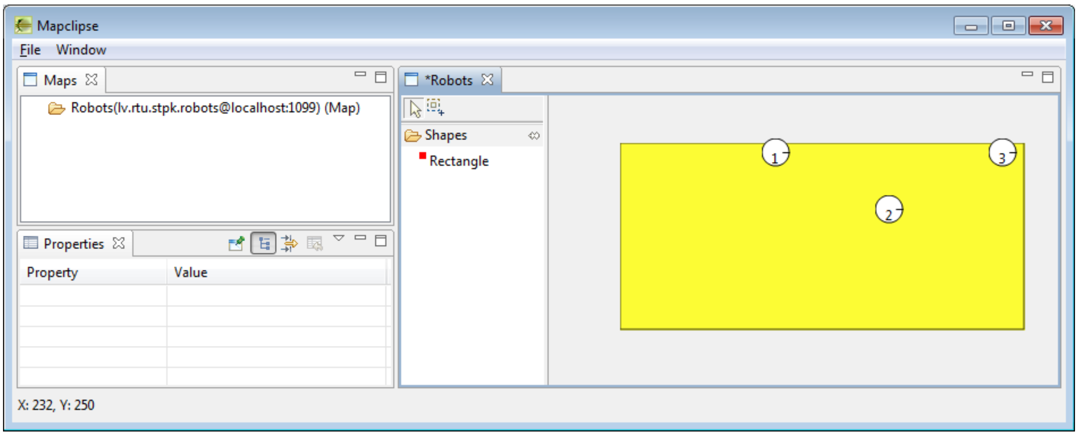


1.7. att. Pievienota karte.

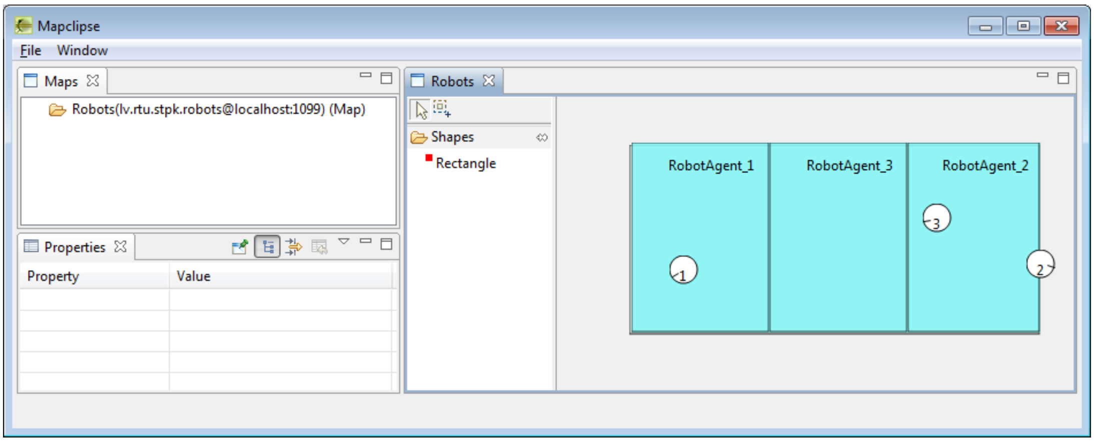


1.8. att. Robotu kartes piemērs.

Robotiem uzdod darbības apgabalu sadaļā „Shapes” izvēloties „Rectangle”, pēc kā kartē ievelk taisnstūrveida apgabalu. Pēc tam saglabā izmaiņas izmatojot [File]→[Save] izvēlni (vai „Ctrl+s” taustiņu kombināciju), tādējādi darbības apgabalu nosūta daudzaģentu sistēmas platformai uzdevuma sadalei starp robotiem (skat. 1.10. att).

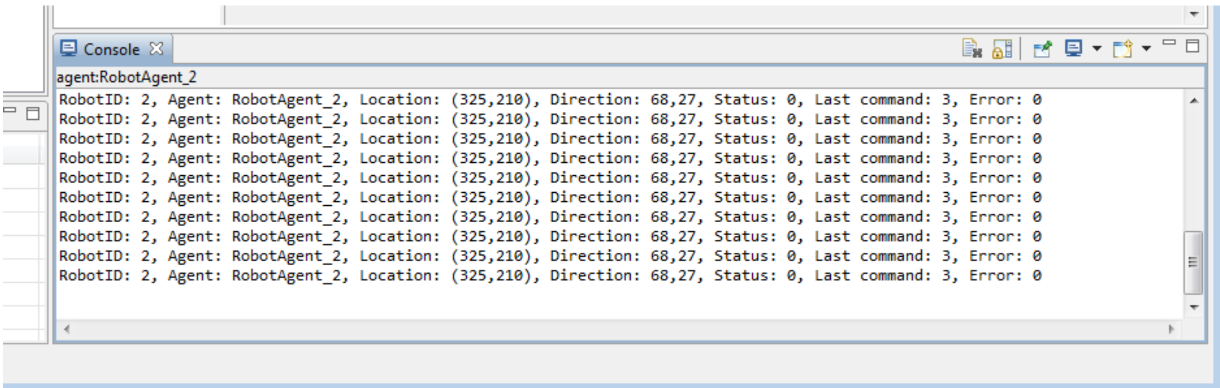


1.9. att. Tikko iezīmēts robotu darbības apgabals.



1.10. att. Robotu atbildības apgabali.

1.10. attēlā ir redzams atbildības apgabalu sadalījums starp robotiem. Lai iegūtu konkrētas robotu koordinātas, ir jāatver konsole, kurā žurnalējas saņemtie robotu stāvokļi. Tādēļ atver [Window]→[Show View…], atzīmē „Console" un apstiprina izvēli ar [OK], pēc kā izkārtojumam tiek pievienots konsoles logs, kurā var redzēt precīzu saņemto aģentu stāvokļu informāciju (skat. 1.11. att) par pašreizējo atrašanās vietu, virzienu, izpildāmo komandu un citiem atribūtiem.



1.11. att. Aģentu saņemto stāvokļu žurnāls.

2. Noslēgums

2.1. Nodevuma pielietojuma apraksts

Daudzaģentu sistēmu var pielietot daudzu robotu sistēmas vadībai. Lai sistēmu pielietotu paredzētajam mērķim, ir jāizstrādā atbilstošs starpniekserveris, kas nodrošina saskarni ar daudzaģentu sistēmu, kā arī saskarni ar robotiem. Izstrādātajā daudzaģentu sistēmā nav specificēta saskarne ar starpniekserveri. Tas nozīmē, ka programmatūras izstrādātājs var pielietot jebkuru saskarni, kas ir savietojama ar izstrādāto daudzaģentu sistēmu, veicot atbilstošu specifikāciju vārtejas aģentā.

Daudzaģentu sistēmu paredzēts izmantot kopā ar izstrādāto lietotāja saskarni. Lietotāja saskarni, savukārt, var izmantot karšu attēlošanai un rediģēšanai. Lai to izdarītu, ir jāveido jauns Eclipse spraudnis. Vienkāršākajā gadījumā pietiek implementēt *BundleActivator* interfeisu, kas nepieciešams spraudņa infrastruktūras inicializācijas laikā.

```
package lv.rtu.stpk.mapclipse.showcase;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;

public class Activator implements BundleActivator {

    public static String PLUGIN_ID = "lv.rtu.stpk.mapclipse.showcase";

    private static BundleContext context;

    static BundleContext getContext() {
        return context;
    }

    public void start(BundleContext bundleContext) throws Exception {
        Activator.context = bundleContext;
    }

    public void stop(BundleContext bundleContext) throws Exception {
        Activator.context = null;
    }
}
```

2.1. att. *Activator* klases minimāls piemērs.

Lai inicializētu statisku datu struktūru, var paplašināt *MapProvider* klasi, kurai norādīt *FinishActionCallback* parametru. Šī parametra objektam pēc veiksmīgas vedņa pabeigšanas tiks izsaukta *performFinish()* metode, kura šajā gadījumā izveido statisku kartes struktūru. Ja ir jāveido

dinamiskas datu struktūras, tad ir jāimplementē datu atjaunošanas struktūras. Jāuzsver, ka papildus atribūtus var pievienot paplašinot *Element* klasi.

Pēc noklusējuma katru slāņa elementu attēlo kā taisnstūri ar uzdotajām dimensijām. Lai veidotu kādas īpašas figūras, *FigureFactory* klasē ir jāreģistrē asociācija starp atbilstošo datu klasi un figūras klasi. Tā piemēram, ja jāreģistrē robota figūra, *FigureFactory* klasei izsauc *registerElementFigure* ar atbilstošā slāņa un parametriem robota klases atribūtiem: *FigureFactory.registerElementFigure(layer, RobotFigure.class)*. Atsauce uz slāni ir vajadzīga, lai varētu nodrošināt atšķirīgas figūras dažādos slāņos pat gadījumos, kad divi dažādi slāņi satur vienas un tās pašas klases elementus.

```
package lv.rtu.stpk.mapclipse.showcase;

import lv.rtu.stpk.mapclipse.extpoints.FinishActionCallback;
import lv.rtu.stpk.mapclipse.extpoints.MapProvider;
import lv.rtu.stpk.mapclipse.model.Element;
import lv.rtu.stpk.mapclipse.model.Layer;
import lv.rtu.stpk.mapclipse.model.Map;
import lv.rtu.stpk.mapclipse.model.ViewModel;

public class ShowcaseMapProvider extends MapProvider {

    public ShowcaseMapProvider() {
        this.setFinishActionCallback(new FinishActionCallback() {

            @Override
            public boolean performFinish() {
                Map map = new Map("Demo", "Demo", 500, 500);
                map.setProviderId(Activator.PLUGIN_ID);
                ViewModel.getInstance().addMap(map);

                Layer layer = map.createLayer("1", "First layer", true);
                Element robot = new Element();
                robot.setId("1");
                robot.setName("Element _ 1");
                robot.setLocation(70, 60);
                robot.setDimension(60, 60);
                layer.addElement(robot);
                return true;
            }
        });
    }
}
```

2.2. att. *MapProvider* klases minimāls piemērs.

2.2. Nodevuma aprobācija

Nodevums aprobēts publikācijās [6] un [5].

Atzinība

Dokumentā aprakstītais pētījums ir finansēts no ERAF projekta „Daudzaģentu robotizētas intelektuālas sistēmas tehnoloģijas izstrāde”, projekta ieviešanas numurs 2010/0258/2DP/2.1.1.1.0/10/APIA/VIAA/005.

Literatūra

[1] FIPA. (2002). *FIPA Contract Net Interaction Protocol Specification* [Online]. Available: http://www.fipa.org/specs/fipa00029/

[2] E. Lavendelis, A. Liekna, A. Nikitenko, A. Grabovskis, and J. Grundspenkis, "Multi-Agent Robotic System Architecture for Effective Task Allocation and Management," in *Proceedings of the 11th WSEAS International Conference on Signal Processing, Robotics and Automation (ISPRA '12)*, UK, Cambridge, 2012.

[3] F. Bellifemine, C. Giovanni, and D. Greenwood, *Developing Multi-Agent Systems with JADE*. Chichester: John Wiley & Sons Ltd, 2004.

[4] JADE. (2010, Sept. 5). *Jade - Java Agent DEvelopment Framework* [Online]. Available: http://jade.tilab.com/

[5] A. Grabovskis, "The generic map visualization framework," *Applied Computer Systems*, vol. 13, pp. 15-21, 2012.

[6] A. Liekna, E. Lavendelis, and A. Grabovskis, "Analysis of Contract NET Protocol in Multi-Robot Task Allocation," *Applied Computer Systems*, vol. 13, pp. 6-14, 2012.