The Generic Map Visualization Framework

Arvids Grabovskis, *Riga Technical University*

***Abstract* – Controlling robots using a multi-agent system is becoming an active research area which includes a lot of challenges regarding localization, management, coordination, task allocation, etc. It is a very important for a human to overview such a system visually – to monitor a robot status, performance and other characteristics. At the same time some concepts relate to all such systems – robots act in certain environment, where there are different kinds of obstacles and objects. In order to fasten initiation of such visualization tools, a new framework is proposed – "Mapclipse" which is based on the popular Eclipse framework. The purpose of the framework is to create a common domain model, API for its management (like reflecting object movements, environment modifications, etc.), a basis for common visualization tasks and provide a means to customize the figures of specific types of objects. This paper describes the concept of the system in detail, the current prototype of framework, some of its extensions, as well as highlights the main challenges in the future.**

***Keywords* – Design patterns, graphical framework, visualization**

I. Introduction

Goodwin in [1] has explored a lot of robotic systems. His research proves a wide use of robotics in different application areas, and graphical user interfaces are very important for such system usability, because it is crucial to overview the state of system, issue commands, measure performance, etc. It is self-evident that graphical user interfaces in the same application area share common features like layout overview, element status information, path tracking, etc. But in spite of common features, each system is built using its own framework.

One map visualization framework offering some common features is provided by JADEX [2] intelligent agent development platform. It uses XML in order to define maps, figures, environment properties and other descriptive information. But since it is tightly coupled with the agent platform, it is not usable if there is another map data source like database or different agent development framework.

The concepts of map composition are basically the same: a map consists of several layers that contain elements. In fact, usage of such common concepts can be adopted not only in robotics, but also in other areas like simulations, simple games, and so on. If element location or other properties can change over the time, then the map also needs to support this dynamic nature, for example in terms of performance – it is very inefficient to redraw the whole map in case of changes.

The main objective is to propose a simplified extensible Java based framework for graphical map environment representation. Framework consists of a domain model, configurable user interface layout and plugin development facilities. The focus is on Java language because the first application area is the controlled robot location visualization of intelligent multi-agent system, where JADE [3] is used as a multi-agent implementation platform.

The paper is organized as follows. In the first section, the reasons of project initiation and of the first designated application area are explained. The second section focuses on exploring different alternatives of visualization frameworks, paying attention to different types of frameworks. The third section describes the intended concepts of graphical user interface and its architecture. The fourth section describes currently developed prototype, as well as the first developed plugin. The sixth section outlines the future work to be done regarding framework development and application areas. Finally in conclusion, the main strengths and weaknesses of framework are outlined.

II. Background

In the year 2010, a new project was initiated in order to develop principles of robot coordination that could enable them to process large areas in terms of low cost infrastructure [4]. Area processing might include cleaning (which is the example application of project), spraying or seeding fields, etc. The specifics of vacuum cleaning robots chosen in the project include that robots have their own algorithm of area processing that should not be broken.

It has been decided that the system will be built as an intelligent multi-agent system where each agent controls its own robot, therefore creating multilevel architecture, where higher level components tend to be more deliberative, while lower level components tend to be more reactive and exhibit swarm-like behavior. Systems of such architecture have been described in [5] and among other robot controlling systems explored in [1].

JADE has been chosen as an agent development framework (the rationale of choosing this alternative is out of the scope of this paper) in order to made Java-based graphical user interface (GUI) development frameworks more favorable. There is a wide variety of such frameworks like Netbeans, Java's built-in Swing, Eclipse, and many others. The decision has been made in favor of Eclipse. That is because of a rich set of ready-to-use components, its extensibility, wide documentation and developer experience.

It is a common practice that the software project development schedule works in parallel on GUI and business logic. Such a schedule involves several problems:

- At early stages of development no real data is available that should be shown in GUI.
- When the first prototype of a working system is ready, it is cumbersome to test and debug GUI features, because of long system startups and tightly coupled and rapidly evolving middle layer of the system.

As a result it has been decided to create a framework to decouple a middle layer (MAS in this particular project) from GUI platform. That would allow extending the application with different plugins for different purposes. For example, at the development time, complex figure customization requires a lot of debugging – a developer must check how the figure is being drawn in case of different states of an object that the figure represents. The intended system should provide a way to write a very simple plugin that suits best while developing different parts of the system. In order to implement an extensible system, several frameworks have been evaluated to find the most appropriate one.

III. Visualization Frameworks

This section focuses mainly on extensible two dimensional map visualization or graphic frameworks for desktop applications. Detailed evaluation of all available frameworks is out of the scope of this paper, hence only some of the best known frameworks are generally reviewed.

One tool that does not fit in any of further categories is the environment visualization tool provided by JADEX in its newest (2.0) version, but evaluating its extensibility and given that a multi-agent platform is going to be implemented into JADE, it turned out that it is very hard to extend the tool outside the scope of simple map drawing. There is no well-defined framework provided in order to modify the map (like drawing new areas) and add custom views other than Java's built-in Swing framework which has a small number of ready complex components (e.g. wizard templates). Another important aspect is that version 2.0 has not been released yet (currently only release candidates are available at [6]) and there is very little documentation available. In fact, the only way to evaluate this framework is to use the example code provided by JADEX.

A. General Purpose Graphical Frameworks

Eclipse Draw2D and Graphical Editing Framework (GEF) [7] are very popular frameworks for developing visual representations for integrated circuits, computer networks, UML diagrams, database structures, etc. GEF is built on Draw2D by implementing different design patterns like model-view-controller (MVC), observer, factory, command, strategy and chain of responsibility in order to improve overall architecture in terms of modularity and extensibility. Other advantages that Eclipse frameworks offer include a rich set of plugins, project maturity and developer experience in work with this framework.

Netbeans community has developed its own framework named "NetBeans Visual Library" [8] as a counterpart of Eclipse Draw2D. It proved to be easy to use at first attempts, but hard to implement more complex features because of lack of documentation and tutorials.

Microsoft Windows Presentation Foundation [9] is another framework for building graphical user interfaces that provides a lot of features and is well documented, but it is intended for use on .Net platform, hence it would require another integration layer in order to connect to JADE runtime.

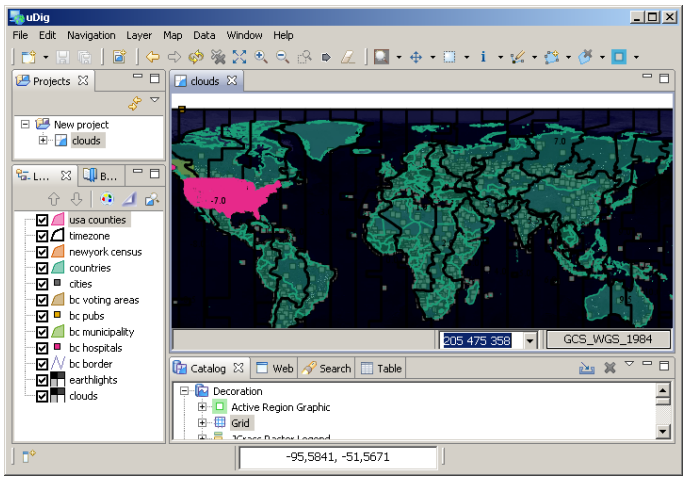

Fig. 1. The user interface of uDig with sample data.

B. Graph Drawing Frameworks

Graph drawing frameworks (otherwise mentioned as toolkits, tools, etc.) are a subset of general purpose graphical frameworks, on which they are usually built. Some examples of such frameworks include Eclipse Zest [10] and jGraph [11]. In fact many of them are built using previously described general purpose graphical frameworks.

Since graph drawing frameworks have specialized in one application area, they can provide a lot of already built services and implement parts of the system that are common to all graph applications.

These frameworks could be usable for map visualization, because all of them provide a means for graph node representation customization that can be used as layer elements. However, a graph domain is not well-suited for such exploits and there is little use of choosing frameworks of this type over general purpose frameworks if the application area is not some kind of a graph.

C. GIS Frameworks

Another active framework development area is geographic information systems (GIS). Resource [12] lists some of the most popular GIS implementations. It states that implementations written in C language are more mature than others because they are being developed for a longer period of time. But this statement is very controversial because of the wide adoption and use cases of Java-based systems.

The system that deserves outlining is uDig [13], which stands for "User-Friendly Desktop Internet GIS". It combines the strengths of GeoTools (design, data, structures, and standards), JUMP (user interface, renderer, and interactivity) and Eclipse (extensibility and industry development standards) projects [12]. The result of this combination is presented in Figure 1, where layers represent different map objects, and the map is rendered using GeoTools as a composition of layers.

Although this and other GIS frameworks can be extended in terms of functionality, they are not the best solutions in case of bounded (e.g. indoor) environment. The reason is that in those kinds of environments concepts of latitude and longitude,

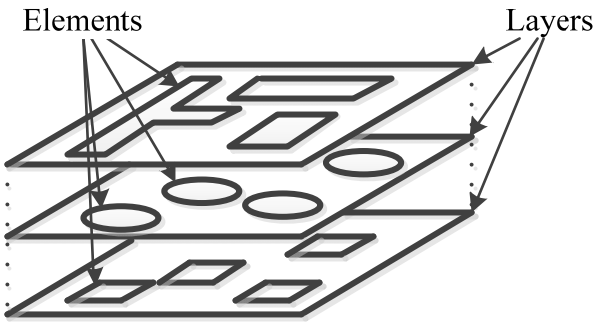

Fig. 2. Map layer composition

which are used as the basis for coordinate system, are irrelevant and might be hard to interpret for a user. Another point is that it requires additional experience regarding GIS development – for example, shapefile definition. Evaluation of uDig leads to a conclusion that GIS frameworks are too complex for application in common areas.

The aim is to find a lightweight tool for easy extension, giving only the very basic functionality. Inspired by graph drawing framework specialization that allowed uniting the used concepts, as well as implementing large part of the system and GIS frameworks, which can be used in the areas where GIS standards are applied, a new framework has been developed.

IV. Introducing Mapclipse

The concept of map composition basically is the same with some minor differences between different areas of application (see Figure 2):

- Map consists of several ordered layers
- Layers contain elements of the same type
- Elements of latter layers overlay the elements of the first ones

The same concepts have been used in GIS applications, but the requirements for knowledge of GIS frameworks and their standards are out of the capabilities of most developers, hence the use of general purpose graphical framework has been chosen. The actual architecture is described in further sections.

A. Intended Mapclipse User Interface

Mapclipse user interface is intended to consist of three main parts: a map list, property viewer and map editor (see Figure 4). Map list (left part of the application window) is a tree viewer component that should contain all maps, their layers and elements. Double-clicking on the map in the list should open the map in the editor. Upon selection of any component, its properties are intended to be shown in the Properties view (lower part of the window) and a corresponding figure in editor is outlined (to the right from the map list).

New maps can be added using wizard where the first step involves map type selection (i.e. plugin that will provide the map), but further steps are plugin-specific and might require server connection parameters (like in case with JADE platform), file locations, etc.

Another important part is a figure customization. Elements in layers must be represented differently and the final figure details are up to exact map provider. Note that in GIS frameworks figures of the same layer have the same design, whereas Mapclipse does not imply such a rule – layers are used in order to provide element grouping, not figure styling.

Map editor is also intended to have customizable element palette. Very simple use-case is the drawing of new figures, but more complex one involves area selection, sending to server, status reflection, etc.

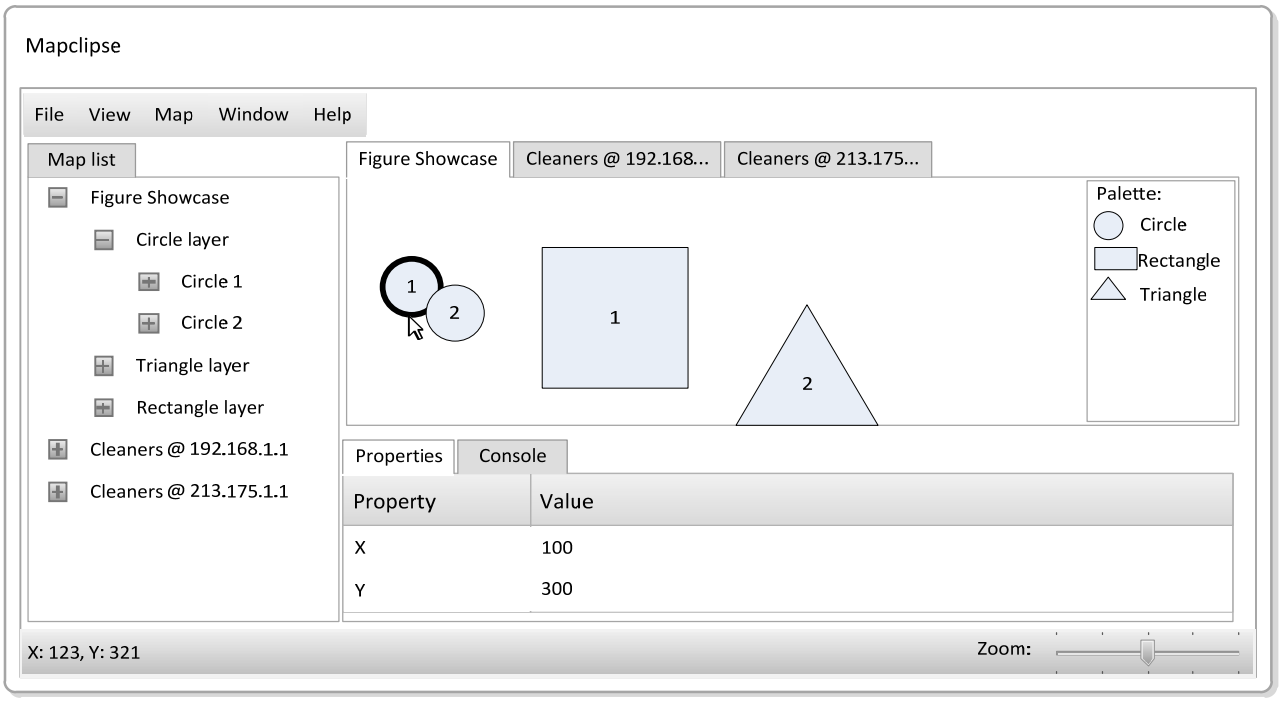

Fig.3. User interface sketch

B. General Architecture Description

Eclipse has been chosen as a base platform for Mapclipse. The choice in favor of Eclipse has been made because of its rich set of tools, good documentation and modularity. In order to build the map viewer, Eclipse GEF has been chosen over Draw2D, on which the GEF is built, because GEF uses MVC design pattern and provides a means to update the view depending on model changes in an efficient way – only those parts of view are updated, which have been changed. Since one of the first Mapclipse plugins exhibits a dynamic map nature, a lot of attention has been dedicated in order to enable map updates. MVC design pattern support this dynamic nature and also makes extending it very easy, because the sole task of plugin would be updating a model and providing view customizations. Note that a controller part of MVC is going to be common for all plugins.

Mapclipse implementation has been split into two parts. The first part is a library that contains classes for domain model, but the second – Mapclipse application that uses model library. The distinction of domain model classes from the rest of the application is very significant because it allows using domain classes without requiring dependencies on large user interface libraries. In this way map data provider implementations can prepare domain objects for presentations natively without adapters. Further the domain model is explained in detail.

C. Domain Model

According to the map structure represented in Figure 2, the class structure shown in Figure 3 has been created. It might seem trivial to implement such a structure, without any needs to go in detail, but the model contains some very important features that are further described and are critical in order to provide change propagation and property descriptor information.

The first important feature of the current model implementation is that it supports model change notifications. This is a standard approach that is used in order to notify other classes (listeners) about changes of the object state. One example of such use case is in GUI, where it must update its contents depending on model changes. GUI updater classes then attach themselves as listeners to the model classes and initiate updates only in case of actual changes instead of model checking on a regular basis or model being responsible for updating GUI.

One problem implementing property change notification has been vague guidelines regarding collection changes. In theory standard PropertyChangeEvent notifications can be propagated, but in this way it is hard for a change listener to identify which collection element has been added or removed. For example, if the listener needs to add or remove elements from the view depending on collection changes, then it must compare all visible elements against the changed collection and then update the view depending on the difference. This operation is expensive in terms of performance, hence – undesired. There are some third party Java libraries like Glazed Lists, which support all collection changes, but it adds another dependency. In order to solve this problem, special CollectionPropertyChangeEvent class has been implemented which supports collection element inserts and removals, and there is sufficient functionality in this case. Using this class, the change event listener can find out the exact removed or inserted element. Since this class extends PropertyChangeEvent, it is compatible with Java built-in property change notification mechanism and can utilize its support classes.

The second important feature is related to Properties view provided as a plugin by Eclipse framework. It can be used in order to view the properties of currently selected object in workbench (according to Eclipse terminology, a workbench can be perceived as the application window) whatever the representation of object is (a figure, list element, etc). Although implementing the data provider for this view is straightforward, it requires a lot of code. In order to simplify this task, the model provides special annotation "Property" which marks class members, getters or setters that should be used as data in Properties view. Property annotation provides a means to specify:

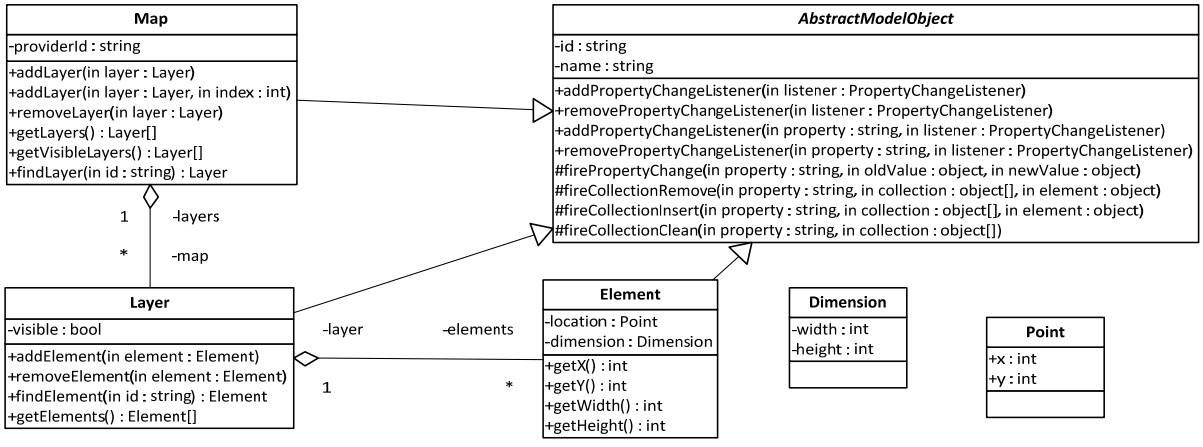

Fig.4. UML diagram of model class structure. NB: some getters and setters of attributes are omitted in order to simplify the figure.

- Property display name – a default text that should be viewed. If a display name is not specified, then the transformed field or method name will be used (field names are capitalized, but "set" and "get" are removed from the field getters and setters).
- Property title key – for use in multilingual environments – in order to resolve an actual display name depending on this title key. If the localized title cannot be resolved, the display name will be used.
- Changeability flag – this is relevant only for class fields and is true in case if the property should be changeable from Properties View.
- Precedence – this determines the order of properties that should be used in Properties Viewer.

This descriptive information is scanned using Java reflection mechanism by Mapclipse application, and Properties view is populated with values according to the object state. For example, the class diagram in Figure 2 shows that "Element" class has "getX" method. The intention of this method is to annotate it with "Property" annotation and thereby showing the "x" coordinate in Properties view.

D. User Interface Mock-Up

Implementation of the map list view is straightforward, because the domain model used by it is well defined. In order to add new maps, plugins must create a new map and add it to a special ViewModel singleton, and the list view will reflect the changes. Map creation is delegated to FinishActionCallback interface implementation that must provide plugins. Interfaces method "performFinish" is called, when the map adding wizard has completed. During this call, plugin must initialize the map and add it to the map list view.

A lot more complex situation is regarding map editor implementation – there is going to be a lot of parts that will differ from one plugin to another. Therefore, default editor implementation only draws figures and is responsible for providing zooming capabilities. The editor base class should be extended by plugins in order to add a map saving functionality or override the default methods if further customizations are required.

GEF uses implementations of "EditPart" class that should be extended in order to control creating, updating and deleting figures in the editor. EditParts is a controller part of MVC design pattern in GEF. Since a common domain model has been defined, then the controller part can be implemented, too. Each implementation of the controller attaches itself to the list of model listeners. For example, a layer controller creates or removes child elements and element controller updates figure location depending on model changes.

The actions executed upon adding a new element in the layer also include associating a figure with the model element. In order to make figures customizable, a special figure registry has been created where each layer is associated with a class of figure implementation. It is up to plugin to register layers provided by them with required figure classes. If no figure class is associated with a layer, then a simple rectangle figure is used.

This user interface implementation provides a lot of default implementations; hence, implementation of FinishAction-Callback is sufficient to view the map.

V. PROTOTYPE

Since the start of the Mapclipse design and development, a lot of work has been done in order to verify its applicability for different domains. It might seem that a vacuum cleaner robot status map plugin has been developed first, but this is not the case – the first plugin has been a simple example plugin created while developing Mapclipse core framework. Purpose of the plugin has been to test and debug different parts of the system.

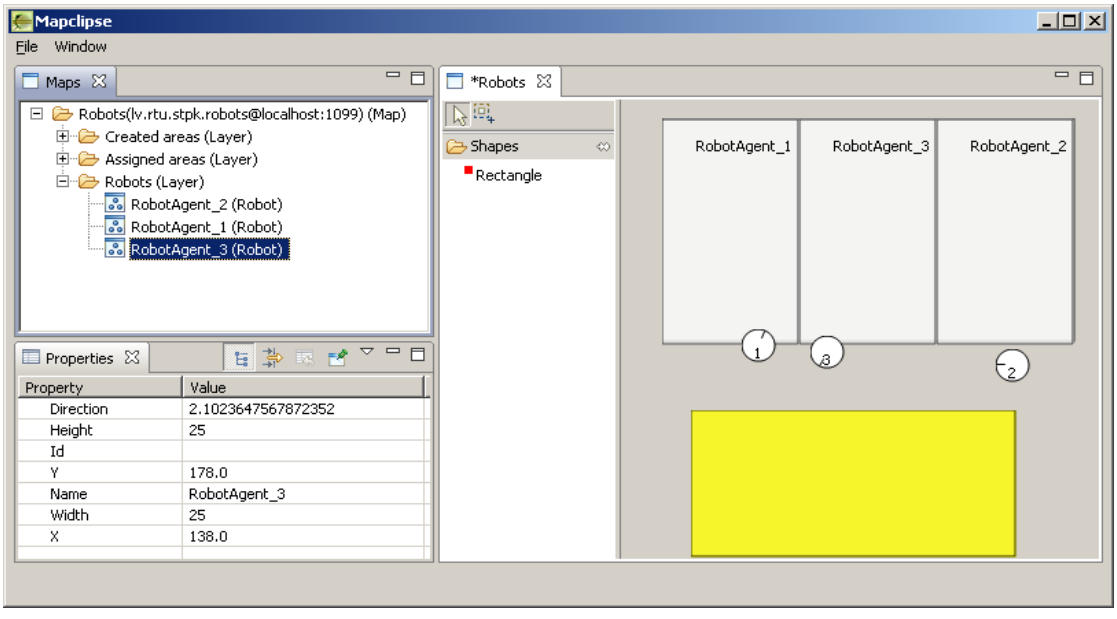

Fig. 5. Mapclipse GUI with vacuum cleaner robot map plugin

This plugin architecture has turned out to be very convenient, because GUI developer can implement, test and debug core framework and plugin mechanism using the example map plugin of which he has a higher degree of control. For example, it has been used to test the behavior of view in case of model updates. Other benefits of using the example plugin include faster system startups during GUI development (no need to run main JADE platform), and focusing merely on GUI development while leaving JADE specifics aside, thus improving developer productivity.

As the second plugin, vacuum cleaner robot map visualization plugin has been developed that is shown in Figure 4. On the left side there is map information – a list of layers and their elements. Below the map list there is Properties view that shows the state of the selected element. In the map editor there can be seen three robots and three areas that are assigned for cleaning. Three adjacent areas are parts of the area created by a user and sent for processing. The lowest area is created by a user, but not yet sent to a multi-agent platform for processing.

In case of vacuum cleaner robots, area processing workflow is as follows:

1. User selects a rectangle drawing tool and draws a rectangle in the map editor area (this functionality is provided by Mapclipse plugin);
2. User saves the map, and plugin sends the coordinates of the area to a multi-agent system for cleaning;
3. Multi-agent system breaks down the area into smaller ones and assigns a part to each robot;
4. Robots process the assigned areas repeatedly reporting their status.

In order to support the workflow, the default editor has been extended with the palette and special save method. Palette contains a single rectangle drawing tool that allows a user to draw an area. Upon saving, the area bounds are sent to JADE platform for processing.

VI. Future work

Further work involves exploring more use cases in order to test the extensibility and completeness of Mapclipse core. Currently there is only one real application area which makes it hard to outline features of general interest and justify inclusion or exclusion of certain features from the core.

One of the future plugin implementation ideas is map data gathered from surveillance cameras and the image recognition system. For example, robots signal that their coordinates are X and Y, but due to self-localization specifics there is no guarantee of those coordinates being precise, hence combining surveillance cameras with the image recognition system and showing that information in GUI might help adjusting a robot self-localization algorithm.

Currently only the domain model library has been completed, but Mapclipse core application is still being rapidly developed and new features can be proposed, but new feature proposals depend on new application areas and wider adoption. One of such a proposed feature is showing a layer of one map into another. Another proposed feature is object movement tracking and reviewing. For example, one is willing to view the path of a robot in order to calculate its performance, but such a task is domain independent, because it is the simple location tracking of layer elements, hence this feature can be implemented in the core framework.

While the domain model is well defined, there is a lot of work to be done in order to improve the quality of practical implementation before the framework can be offered to general audience.

VII. Conclusions

In this paper, the author has proposed a generic map visualization framework “Mapclipse” which is based on Java platform. The prototype is implemented using Eclipse Graphical Editing Framework. Wide usage of different design patterns enables further extension of Mapclipse in terms of functionality and view customization.

The defined domain model provides a means to extend the framework for a very wide range of applications while the dynamic model change propagation requires very little effort to implement the first system prototypes.

Extensibility of the framework has simplified the development process in terms of testing and debugging, because the developing base GUI has been done by implementing very simple example plugins that exercised all possible extension points.

The first real field of application is visualizing the location of vacuum cleaning robots in their environment where robots are being controlled by a multi-agent system. It has been proved that the plugin development is simplified, because the custom complex figure testing and debugging can be performed by implementing a very simple example plugin that uses the figures that are being developed, thereby not requiring any running server, multi-agent platform or other heavy components.

Acknowledgements

The research has been partly supported by ERAF European Regional Development Fund project 2010/0258/2DP/2.1.1.1.0/10/APIA/VIAA/005 “Development of Intelligent Multiagent Robotics System Technology”.

References

[1] J.R. Goodwin, "An Unified Design Framework for Mobile Robot Systems," PhD thesis, University of the West of England, 2008.

[2] A. Pokahr, L. Braubach, and W. Lamersdorf, “Jadex: A BDI reasoning engine,” Multi-Agent Programming, pp. 149–174, 2005.

[3] B. Luigi, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems with JADE*. Wiley, 2007.

[4] Riga Technical University, "Informācija par projektu," February 2011. [Online]. Available: http://www.rtu.lv/content/view/5226/1867/lang,lv/

[5] A. Orebäck andH.E. Christensen,"Evaluation of architectures for mobile robotics," Autonomous robots, vol. 14, pp. 33-49, Jan. 2003.

[6] "Jadex BDI Agent System". [Online]. Available: http://sourceforge.net/projects/jadex/

[7] IBM Redbooks, Eclipse Development Using the Graphical Editing Framework And the Eclipse Modeling Framework. IBM,

[8] J. Petri, NetBeans Platform 6.9 Developer's Guide. Packt Publishing, 2010.

[9] D. Chappel, "Introducing Windows Presentation Foundation," September 2006. [Online]. Available: http://msdn.microsoft.com/en-us/library/aa663364.aspx
[10] The Eclipse Foundation, "Zest". [Online]. Available: http://www.eclipse.org/gef/zest/
[11] "jGraph". [Online]. Available: http://www.jgraph.com/jgraph.html
[12] P. Ramsey, "The State of Open Source GIS," presented at 2007 Free and Open Source Software for Geospatial, Victoria, Canada, 2007.
[13] "uDig : Home". [Online]. Available: http://udig.refractions.net/

Arvīds Grabovskisis, PhD student at Riga Technical University. He received his Bc.sc.ing degree in 2008 and his Mg.sc.ing. degree in 2010 by graduating the same university. His major field of study is computer science, particularly computer systems.

He currently works at Riga Technical University as a Software Developer and he is the Developer Group Leader. In addition he is a Research Assistant at Riga Technical University. His research interests are modular system design and development, artificial intelligence, knowledge representation using logic, intelligent agents and their development.