6

Software Agent Finds Its Way in the Changing Environment

Algirdas Sokas
Vilnius Gediminas Technical University, Department of Engineering Graphics
Lithuania

1. Introduction

Currently growing popularity of artificial intelligence technologies has evolved agent technology to develop intelligent agents. The scope of agents' use is very broad. Intelligent agents are software programs designed to act autonomously and adaptively to achieve goals defined by their human developers. These systems make use of a knowledge base and algorithms to carry out their responsibilities (Haynes et al., 2009). In the opinion of Russel and Norvig (2009), an agent is just something that perceives and acts. Dependent on their roles, skills and environment, an agent has his own capacity. In the opinion of Nwana (1996), an agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment and by doing so, realize a set of goals or tasks for which they are designed.

Concept of agents leads to the early years of investigation of distributed artificial intelligence - the 1970s, particularly with Carl Hewitt "Actor" model. Hewitt suggested this model was independent, interactive and parallel treating object, which he called the "actor". The object was able to acquire some of the internal states and could respond to messages of similar objects. Actor - a computational agent, which has an e-mail address and a behavior. Characters communicate by sending messages and support their actions acting jointly, (Hewitt, 1977).

Nwana, in order to carry out a review, split all field studies of agents out into two directions: one research stream from 1977 until the date of his research (1996), the other from 1990 until the same date of the research (1996). The first one examined the direction of smart agents. This era began in the late seventies and concentrated mainly on deliberative-type agents with simple internal action models. Subsequently, Nwana identified these agents as cooperating agents. Advisory type agent is the one which has a clearly expressed symbolic world model and its decisions (for example, about what actions to perform) are based on symbolic grounds (Wooldridge, 1995).

At the beginning, the first direction focused on macro issues such as interaction and communication among agents, task decomposition, coordination and cooperation, and conflict resolution through negotiation and so on. The main objective was to define, analyze, and integrate (used in) systems, which consist of several cooperating agents. These studies have produced results in the classical systems and works, such as the "actor" model (Hewitt, 1977), MCS (Doran et al., 1991), the coordination of network activities. These agents' macro characteristics, as called by Gasser (1991), highlighted the advantages of Society of agents over the Individual agents, which are associated with micro issues. These issues are well summarized in (Chaib-draa 1992, Gasser et al., 1995) works.

In addition to the macro-level questions, the first direction of research can be also divided into the theoretical, architectural and language problems. Work related to these topics occurred naturally by exploring the macro questions. It has a very good overview by Wooldridge & Jennings (1995a) work, as well as Wooldridge & Jennings (1995b) and Wooldridge et al. (1996).

Since 1990 other software agent research and development policies clearly formed and led to a significant variety of the software agents types, which are currently being investigated. Nwana (1996) extended Wooldridge & Jennings (1995a, b) works by observing what was not included in the past era scientists work and expanded the list of the types of agents researched. Nwana states that the types and classes of agents need to be explored in addition to examining the macro topics and theory, architecture and language matters.

The agent acts independently. It is not a called component; it is an active, monitoring its environment and responsive entity. The agent monitors its environment and is able to respond to changes in a manner to be able to continue to pursue the objectives of the task. The most common goals of one agent may be narrower, so then several interacting agents are needed in order to achieve the objectives of the interest to a person or place the necessary processes. Agents can cooperate in working towards a common goal, or simply interact with each other, each to its own objectives. Exclusive multi-agents system feature is a potential opportunity to provide its global objectives. Intelligent agent is a program that reacts to the sensations of certain actions (Russel & Norvig 2009). Agent "feels" the environment and decides what actions are adequate to his senses and acts on them. Agents operate independently or nearly independently as a communication link between users and other software systems. Agents use following features: the continuity of time, autonomy, sociability, rationality, and ability to respond to the environment, adaptation. According to the architecture of agents are:

- Logic based agents. Such agents decide on action to be taken shall logical deduction method. Agent tries to make the need for action, using deductive proof. The agent is trying to prove process using deductive proof. If the formula is proved, the agent performs the act.
- Reactive based agents. These agents are simply responsive to the environment, but not reading mechanism.
- Belief-desire-intention agents. Such agents are within the beliefs, desires and intentions, and their decisions according to these three things impressions. Filtering function updates the agent's intention in accordance with its beliefs, desires and intentions of the current. Finally, the action selection function selects the most appropriate action according to the agent's intentions.
- Layers based agents. These agents take decisions during the software architectural layers. Each layer fulfill of the different levels of abstraction. There may be vertical and horizontal layers architecture. Behavior of each layer can be treated as a single agent's behavior.

This article analyzes software agent in the changing environment. Find shortest way between two points in the flat space with prominent polygon fences. This is an idealized task that a robot (agent) has to solve seeking to find its way in the environment (drawing).

The article example based on previously created technology (Sokas 2005, 2010). Graphical system can analyze drawing, forming graph, calculate graph matrices, extract route and prepare programs form with information. It discerns objects-classes: agent, graph, route, which have some properties and methods. System test is executed with drawing. Design system and example of programming agent in the drawing is presented. The creation tasks of programming agent system are solved with Agent Unified Modeling Language (AUML).

2. AUML for modeling intelligent system

Automated programming system designers use object-oriented design methods. Based on this, Unified Modeling Language (UML) was created, which is standard for describing system structure and principles of working (Rumbaugh et al., 1999). The AUML is used for designing varied programs and systems with using agent technology (Odell et al., 2000). Modeling language AUML is still being advanced today (Corchado et al., 2008; Xiao, 2009; Bajo et al., 2009; Vallejo et al., 2011).

Design system may be approached as a group of objects which members use common efforts trying to realize particular functionality. We begin to research what objects are needed for every task of user case diagrams and how these objects interact among each other.

Begin to analyze programming agent working in the drawing and collaboration of systems objects. Use case diagram for designing such type of agent has following cases: analysis of environment, graph formation, search execution, shortest way extraction and showing in the drawing. Begin to analyze only case of searching for shortest way between two points shown in the drawing. Collaboration diagram describes objects behavior in one user case zone. If user case diagrams describe the system at the end-user level, then collaboration diagram presents realization elements such as a class, objects and relationship among them. Collaboration diagram describes collection of objects, which in special situations work as united ensemble. The diagram presents ensemble's static (connections that link objects) and actions (sending messages). It accents the static ensemble structure. The messages in collaboration diagrams are numbered for showing the sending order. Collaboration diagram describes particular situation and is useful to present objective range analysis results, but is limited because we can show few messages in the diagram. Designed system user case "analysis of a drawing, graph formation, search execution, shortest way extraction and showing in the drawing" is presented in collaboration diagram (Fig. 1.).

In this collaboration diagram user controls a drawing and a form. After changing any coordinate of object point, programming agent begins to self-operate. The agent automatically analyses the drawing, forms a graph with nodes that are object points, calculates all paths from start to destination node and extracts shortest route. After that the system automatically shows this route in the graph and draws the route in the drawing and presents a form with the information.

State chart diagram describes objects' dynamic behavior only in one class (Dunn-Davies, 2005; Pan, 2009). State changes may happen because of inside transformations and actions from outside objects. We have not designed classes yet but of course a class can be a agent in the drawing. Lets analyze formation of a agent dynamics (Fig. 2.).

A software agent has three states. The first state is the analysis of a drawing, where current objects' coordinates are checked against the original ones. The first event that triggers the

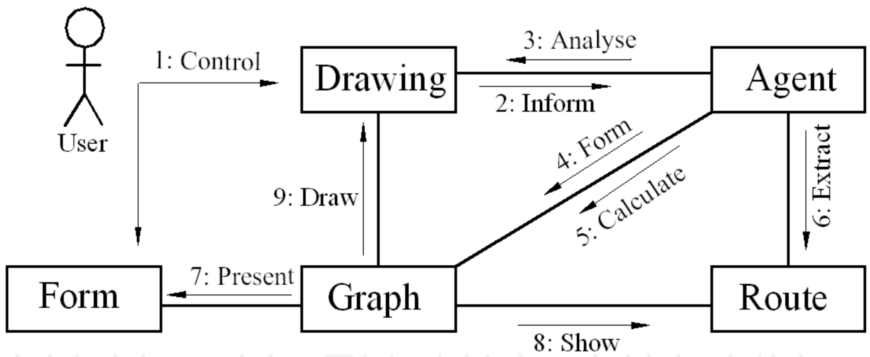


Fig. 1. System collaboration diagram

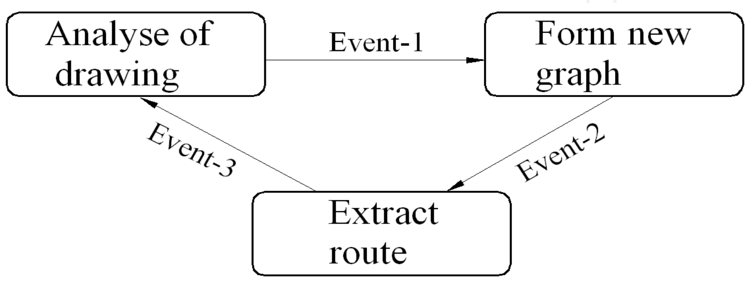


Fig. 2. Agent object statechart diagram

second state is the coordinates change of the drawing's objects. The second state is the forming of a new graph with drawing's objects. The second event that creates the third state is the new graph in the drawing. The third state is the extraction of the shortest route from the first to the end node. The third event returns to the first state but with the changed drawing and the found shortest route.

The class diagram presents static structure of a system. The agent in the drawing is composed from generalization links connected classes: agent, graph and route (Fig. 3).

All messages from collaboration diagram example for the object agent (analysis, form, calculate, extract) are presented as class operations. We will analyze class object operations in the next chapter.

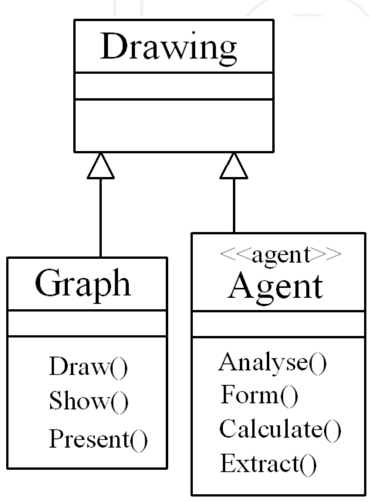


Fig. 3. System class diagram

3. Graphical objects and information

Object-oriented programming greatly facilitates a programmer's work because task is divided, as you can see from Fig 3, into two parts. Create class agent from object agent in the class diagram. All operation (analysis, form, calculate, extract) are programmed as class procedures. In this way class procedures become class methods, common variables – class properties. Graphical system AutoCAD is widely used in the world because of its open architecture and many system files that can be understood by programmers. In the system's environment, a user can operate other programming languages using standard drawing and modeling commands, creating own functions. In the AutoCAD environment we can program with Visual Basic for Applications language (VBA) (Cottingham, 2001; Sutphin, 2006).

Following operations are made by programming method: first, analyzing the drawing and identifying graphical objects. Second, all points of the drawing objects are given graph nodes assigned with numbers and the nodes are connected with edges. Third, calculating prepared matrices and getting the shortest way between all nodes of the graph. Forth, extracting the shortest path between the start and end nodes of the graph. Fifth, showing the shortest path in the graph and, sixth, drawing the shortest path. Objects in the drawing are formed of lines and present prominent polygons. Among objects are intervals. This way we can write all lines of drawing start and end point coordinates into objects matrix [*ObjM*] with method *Analyse_Drawing*:

```
For i = 0 To sk - 1
   Set obj = ThisDrawing.ModelSpace.Item(i)
   ObjM(i + 1, 1) = obj.StartPoint(0)
   ObjM(i + 1, 2) = obj.StartPoint(1)
   ObjM(i + 1, 3) = obj.StartPoint(2)
   ObjM(i + 1, 4) = obj.EndPoint(0)
   ObjM(i + 1, 5) = obj.EndPoint(1)
   ObjM(i + 1, 6) = obj.EndPoint(2)
Next
```

All information about drawing in dxf format, which is used in many graphical systems, will be studied. The data describing the entity is a list. It is made of different dxf group codes. Each such group separated by brackets also forms a list from code, dot and meaning. Code defines property, dot is a distinctive sign, and meaning is the parameter of property. For example, a list (0. "CIRCLE") informs that the code equals to zero and defines entity type, meaning is entity name. Code "-3" means that the next long list is a user extended data (Fig. 4). Additional data named extended data (xdata) may be appended to the graphical entities (Autodesk ,2001).

The next procedure are drawing graphical object the circle and creating the extended data of new graphical object. Information is named "Node". There are three extended data: number of object, circle center x coordinate and circle center y coordinate. Integer and real values are attached with codes "1070" and "1040":

```
Dim ObjCircle As AcadEntity
Dim CenterCircle(0 To 2) As Double
Dim Code1(0 To 3) As Integer
Dim Value1(0 To 3) As Variant
Set ObjCircle = ThisDrawing.ModelSpace.AddCircle(CenterCircle, 2)
```

```
Code1(0) = 1001: Value1(0) = "Node"
Code1(1) = 1070: Value1(1) = j
Code1(2) = 1040: Value1(2) = c(0)
Code1(3) = 1040: Value1(3) = c(1)
ObjCircle.SetXData Code1, Value1
ObjCircle.Update
```

Each graph node stores information about its number and coordinates. After a change in the graph, a transfer of a node to another location, information is automatically updated. Agent examines the changes and compares them with the original graph node matrix [*CM*]:

```
i=1
For Each ObjEntity In ListSelection
    ObjEntity.GetXData "Node", Code2, Duom2
     If CM(50 - i, 1) = Value2(1) Then
     If CM(50 - i, 2) <> Value2(2) Or CM(50 - i, 3) <> Value2(3) Then
          CM(50 - i, 2) = Value2(2)
          CM(50 - i, 3) = Value2(3)
      End If
      End If
     i = i + 1
Next ObjEntity
```

When the agent finds a change in coordinates of the graph nodes, it begins to operate, finding the shortest route from the initial to the end node.

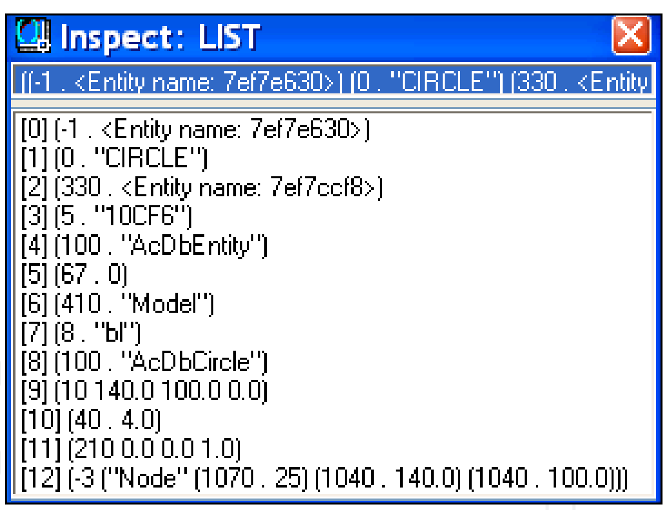


Fig. 4. Drawing interchange format for graphical object circle with extended data

4. Shortest route modeling with graph

In literature the mathematical notion graph presented as figure formed from points (or nodes, or vertex) and lines (or edges, or arcs). A graph is a pair $G = (N, E)$ of sets, where N – set of nodes, which number n is order of graph; E – set of edges, which number m is dimension of graph (Diestel, 2000). The problem is presented as non-directional graph, where edges are without directions. The path in the graph is set of nodes. The route length is equal sum of path edges lengths. Each node has some information as drawing point (coordinates). All nodes are connected by lines. These lines are shown as graph edges. Each edge also has information as drawing line (length, angle, start and end point coordinates). Literature presents several algorithms which find shortest way between two points from concrete graph node to all the other ones. They are Dijkstra, Bellman – Ford, Johnson, Floyd – Warshall algorithms. Floyd – Warshall algorithm is the simplest and fastest (Cormen et al., 2001). Floyd – Warshall algorithm is selected for finding shortest way from one graph node to another selected node. The algorithm uses intermediate node idea. It approaches path among all intermediate nodes and finds shortest route. Foundation of the algorithm is recurrent formula (1), where $d_{ij}^{(k)}$ is the shortest distance from node i to node j with intermediate node from set k =1, 2,...,n. If intermediate node is absent on the way, then the shortest distance is equal to the length of the way, or if k = 0, that $d_{ij}^{(0)} = w_{ij}$. Result of the algorithm is two symmetric and quadratic matrices n measurements: shortest way distance [*DM*] and intermediate nodes [*PM*] matrices. Matrix [*PM*] is filled in this way: if node k is on the way between i and j, then its index equals p_{ij}.

$$d_{ij}^{(k)} = \begin{cases} w_{ij}, & k = 0, \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right), & k \geq 1. \end{cases} \tag{1}$$

Polygon points become nodes of the graph, and lines become edges of the graph. The method *Form_Graph* forms a graph dependent on number of nodes n, nodes matrices [*NodM*] $(n \times 3)$ and it presents edges matrices [*EdgeM*] $(m \times 3)$.

For calculating the graph two matrix $(n \times n)$ dimensions are presented: [*DM*] – distance between nodes and [*PM*] - intermediate nodes. The distances are determined from [*EdgeM*] with method *Prepare_Matrices:*

```
Public Sub Prepare_Matrices(n As Integer, m As Integer, EdgeM)
  For i = 1 To n
    For j = 1 To n
      If i = j Then
        DM(i, j) = 0
      Else
        DM(i, j) = 9999
      End If
      PM(i, j) = 0
    Next j
  Next i
  For i = 1 To m
    DM(EdgeM(i, 1), EdgeM(i, 2)) = EdgeM(i, 3)
    DM(EdgeM(i, 2), EdgeM(i, 1)) = EdgeM(i, 3)
  Next i
End Sub
```

Later Floyd – Warshall algorithm is used which fills matrices [*DM*] and [*PM*] with method *Calculate_Matrices*:

```
Public Sub Calculate_Matrices()
For k = 1 To n
  For i = 1 To n
```

```
            For j = 1 To n
                If (DM(i, k)+ DM(k, j) < DM(i, j)) Then
                    DM(i, j)= DM(i, k) + DM (k, j)
                    PMat(i, j) = k
                End If
            Next j
        Next i
    Next k
    End Sub
```

The method *Extract_Route* determines the shortest path. Method *Present Form* presents program form with calculation results and matrices.

Algorithm of shortest way between two nodes. In the cycle from graph first node until end node use method *Extract Route* which realizes following procedure:

```
Public Sub ExtractRoute (sp As Integer, ep As Integer)
  rl = DM(sp, ep)
  rs = 1
  RP(0) = sp
  RP(1) = ep
  FindPath sp, ep
End Sub
```

There *sp* – start point index, *ep* – end point index, *rl* – route length, *rs* – route size, *DM* - distance matrix, *RP* – route points vector.

Method *Find Path* finds shortest distance among start and end nodes. The algorithm of this method presented in Figure 5.

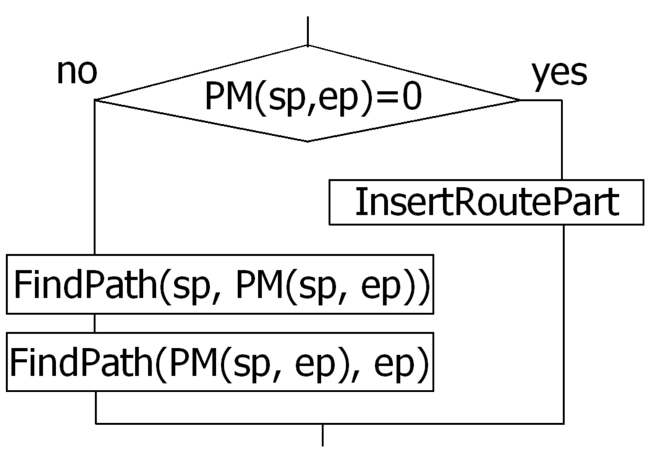


Fig. 5. Algorithm of method *Find Path*, there *PM* - path matrix, *sp* – start point index, *ep* – end point index.

The method *Find Path* which realizes following procedure:

```
Private Sub FindPath(sp As Integer, ep As Integer)
  If PM (sp, ep) = 0 Then
        InsertRoutePart sp, ep
  Else
```

FindPath sp, PM(sp, ep)
FindPath PM(sp, ep), ep
End If
End Sub

This procedure applies the recursion that is a method for the ability to call itself. This is particularly reduces the volume of program code. In recursion programs necessary to provide an output from the program, otherwise the program "hung". If the procedure calls the other procedure, it suspended its execution until the called procedure is carried out. It performs command *Exit Sub*, which applies to the next procedure.

The method *Insert Route Part* realizes following algorithm presented in Figure 6. Part of algorithm Block presented in Figure 7.

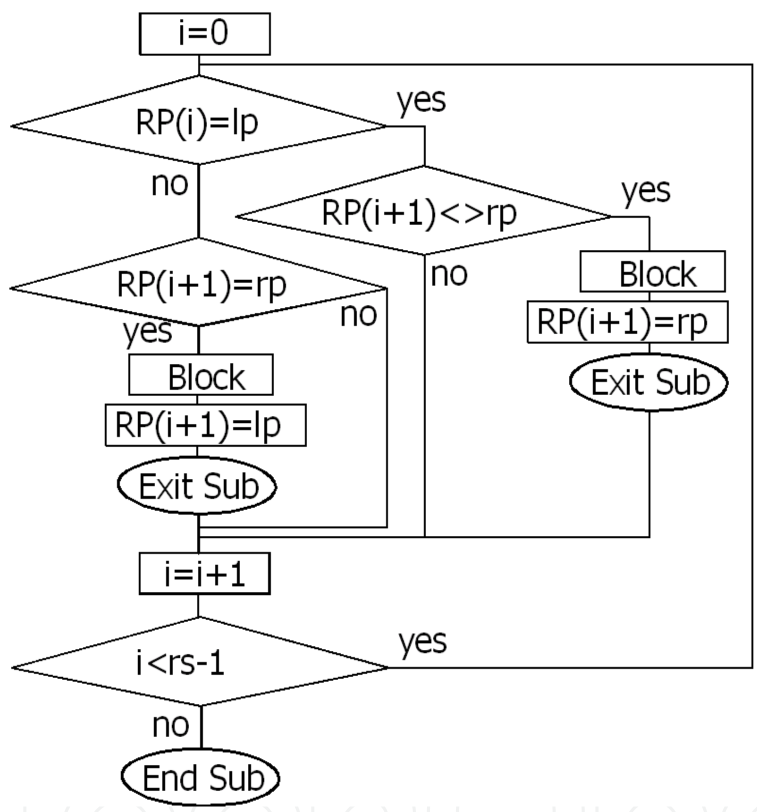


Fig. 6. The algorithm of method *Insert Route Part*

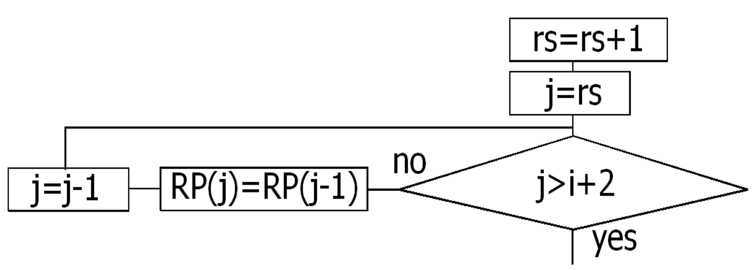


Fig. 7. Part of algorithm *Block*

The method *Insert Route Part* realizes following procedure:

```
Private Sub InsertRoutePart(lp As Integer, rp As Integer)
    For i = 0 To rs - 1
        If RP(i) = lp Then
            If RP(i + 1) <> rp Then
                rs = rs + 1
                For j = rs To i + 2 Step -1
                    RP(j) = RP(j - 1)
                Next j
                RP(i + 1) = rp
            End If
            Exit Sub
        ElseIf RP(i + 1) = rp Then
            rs = rs + 1
            For j = rs To i + 2 Step -1
                RP(j) = RP(j - 1)
            Next j
            RP(i + 1) = lp
            Exit Sub
        End If
    Next i
End Sub
```

There *lp* – left point index, *rp* – right point index, *rl* – route length, *rs* – route size, *RP* – route points vector, *i* and *j* circle indices.

5. Example

Drawing (Fig. 8) has twelve rectangles plane figures. It also indicates two extra points: 1 (start) and 50 (end). The program form (Fig. 9) shows vertices matrix, which has number column and two x and y coordinates columns. The form shows edges matrix, which first column presents edge start point numbers, second presents edge end point numbers, and the third column presents lengths of edges. The edge 2-3 is equal to 80 mm and 3-4 is equal to 30 mm, which we can see from the drawing grid, which is equal to 10 mm. From vertices and edges matrices using programming method I created 50 nodes and 105 edges graph (Fig. 10). After changing coordinates for any node, agent begins to solve the task – to find the shortest path from the first to the end node. Floyd-Warshall algorithm, for finding shortest path from one graph node to another, presents two matrices (Fig. 9). The path matrix shows from which intermediate node from node to node is the shortest way. From node 1 (first row) to node 6 (sixth column) intermediate node is 5. The length matrix presents minimum distances between concrete nodes. From node 2 (second row) to node 4 (forth column) minimum distance is 30+80=110 mm. Using path matrix the shortest path from start node 1 to end node 50 of the graph is found. The answer is in the program form (Fig. 9) top left corner: path distance and set of nodes. It is also presented in the drawings. The second drawing (Fig. 11) has changed only node 6 horizontal position by 63 mm, but result is different – another shorter path is found and distance is 439 mm. The third drawing (Fig. 12) has also changed node 6 and node 25 horizontal positions and the result is totally different – a third shorter path is found and distance is 435 mm. The fourth drawing (Fig. 13) has also changed nodes 6, 25 and 45 horizontal positions and the result is totally different – a fourth shorter path is found and distance is 424 mm.

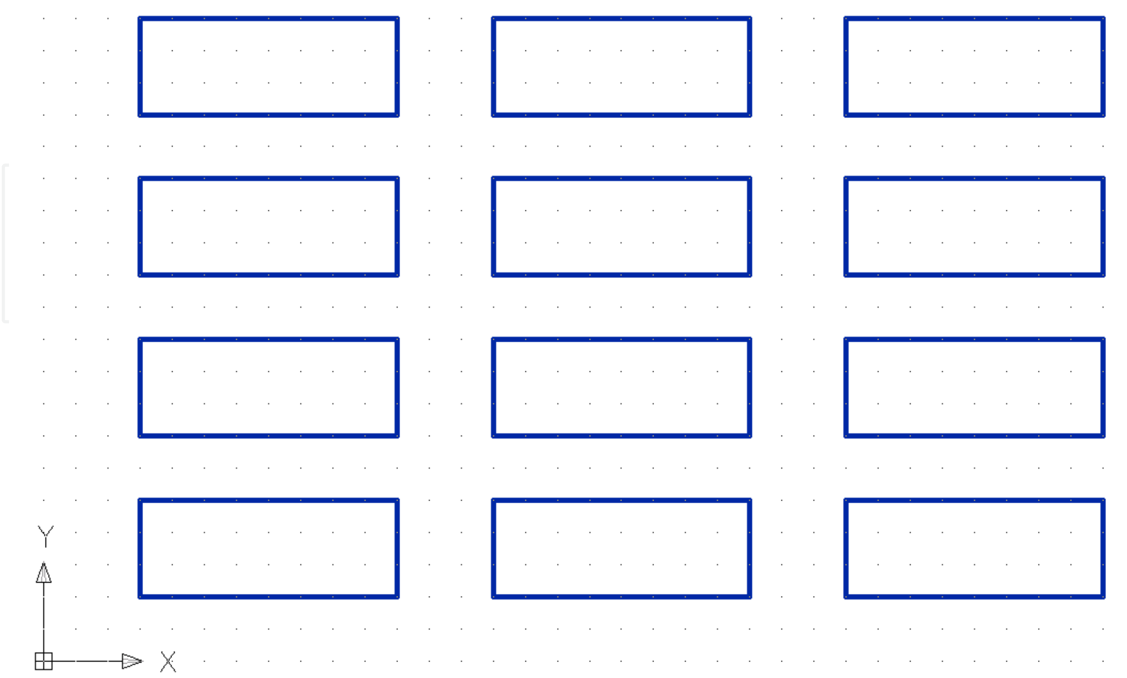


Fig. 8. The environment drawing

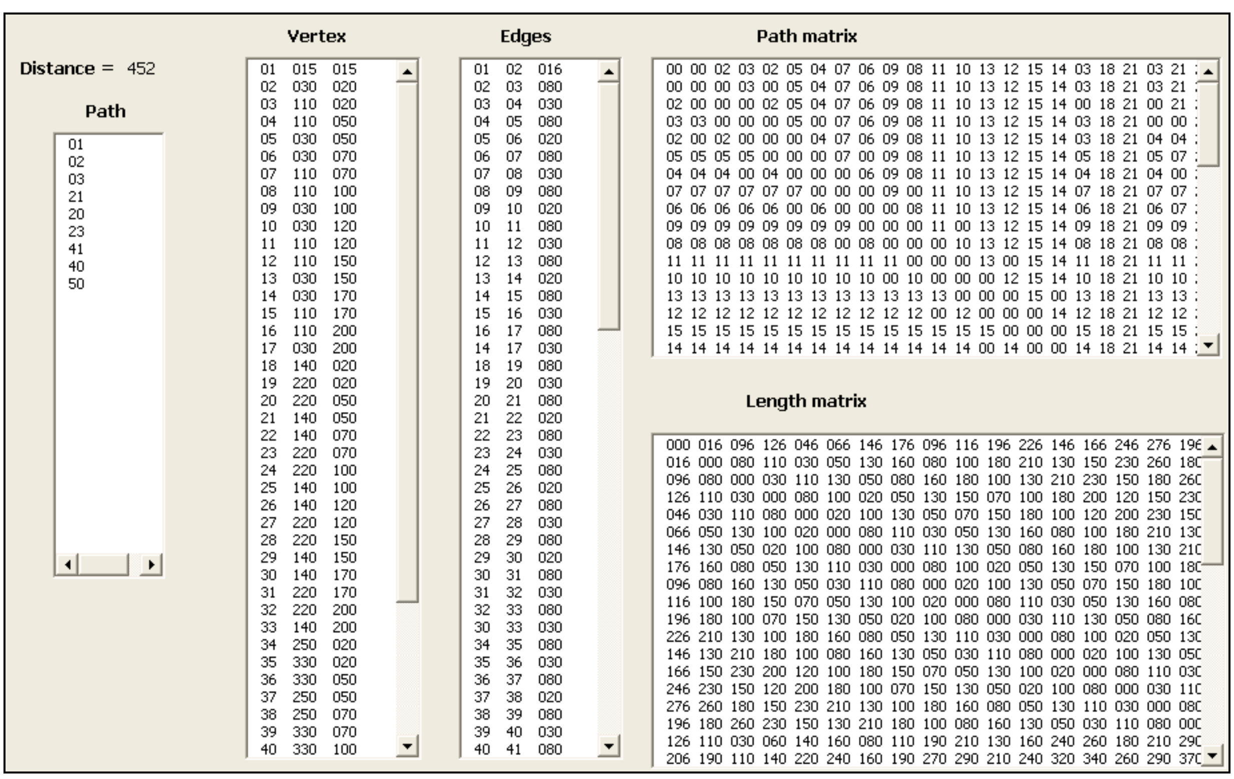


Fig. 9. The program form

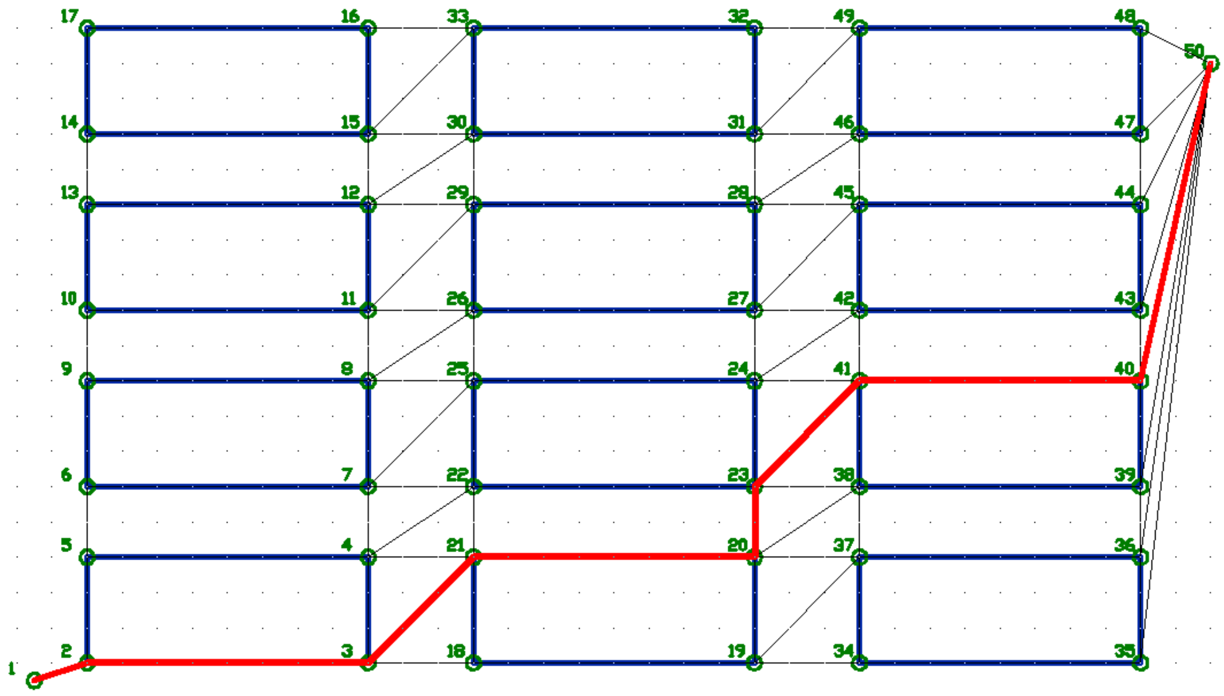


Fig. 10. First drawing

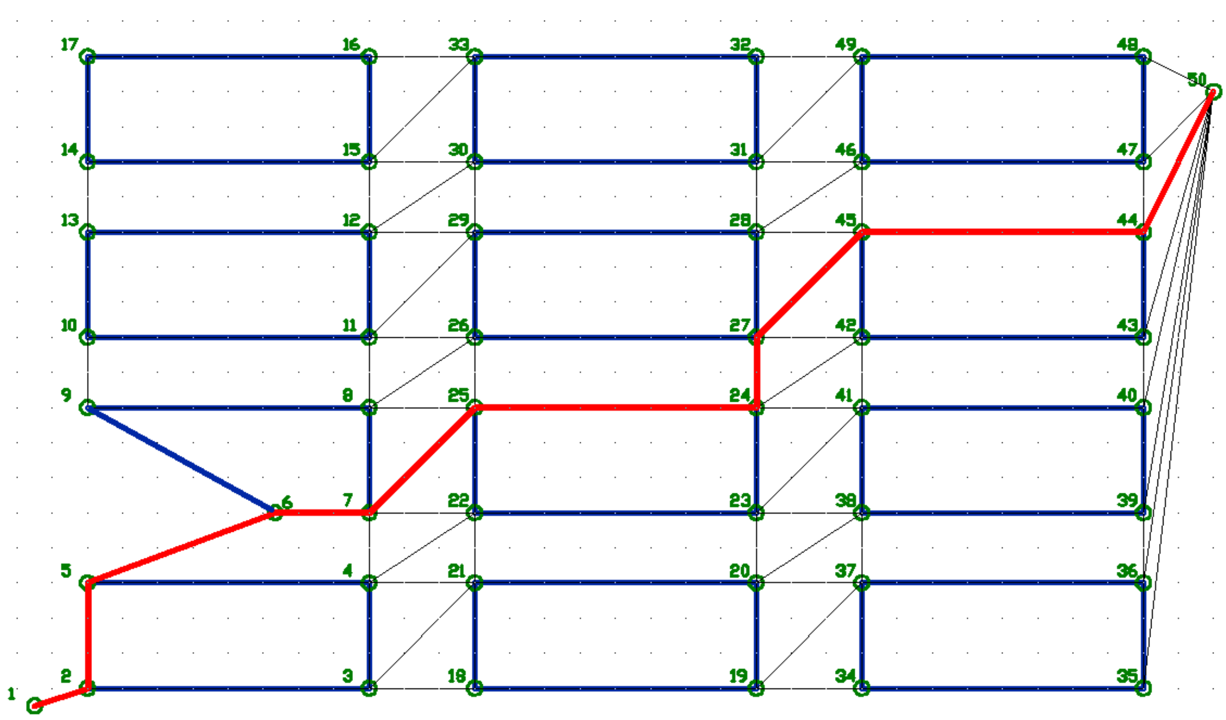


Fig. 11. Second drawing

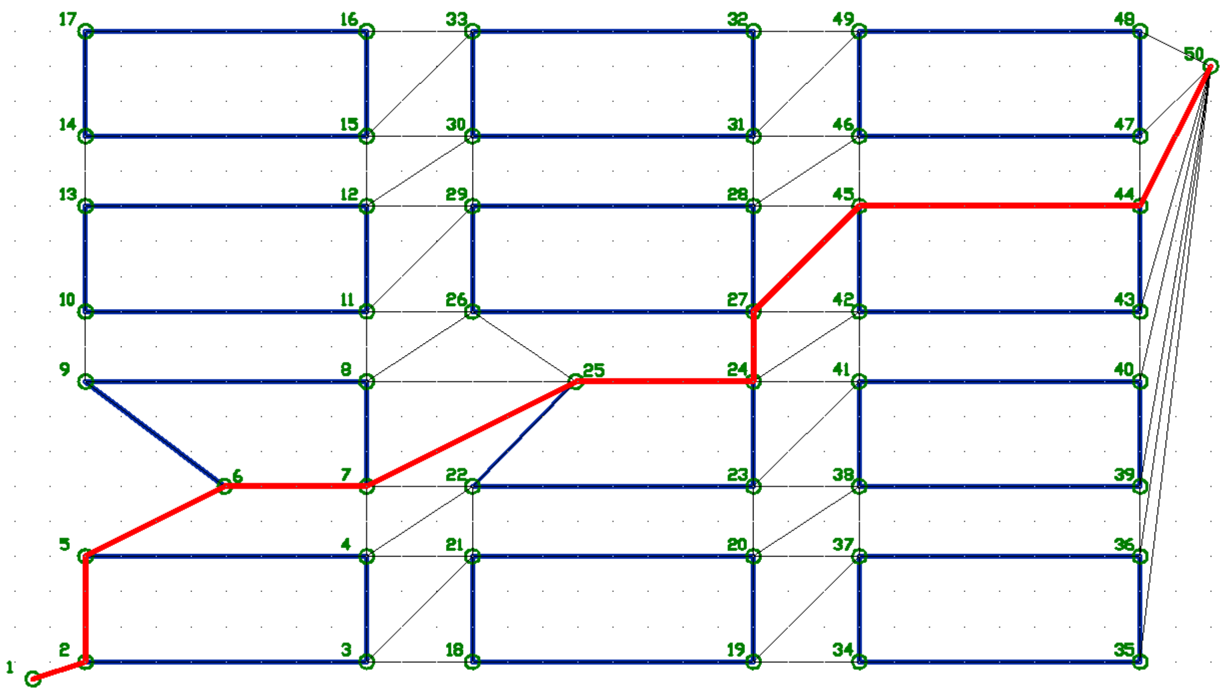


Fig. 12. Third drawing

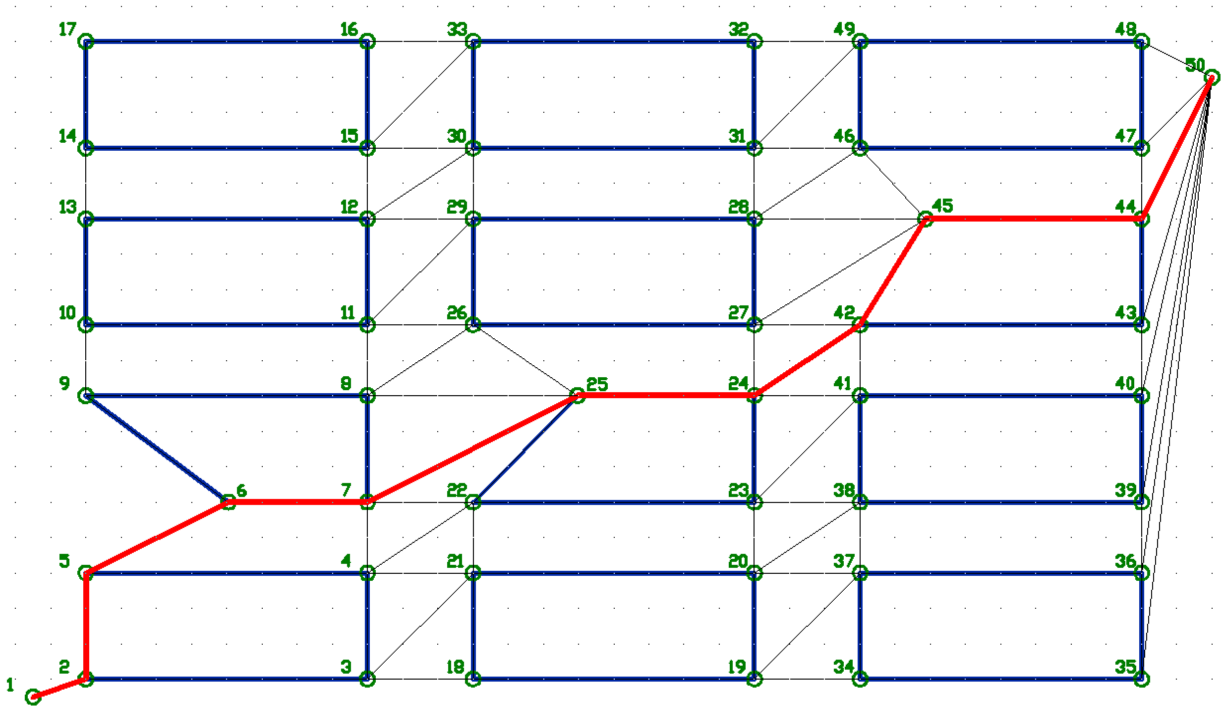


Fig. 13. Fourth drawing

6. Summary

Literature analysis indicates that software agents are used for solving different tasks and that agents are wonderful programming tool for users in the industry and science. The system is modeled by AUML. Presented project demonstrates system object classes as well as their methods and properties. It can design to individual variables activity diagrams. All of this makes programmer's work and communication with customers much easier. The object-oriented programming language, which directly allows implementation of AUML project, is used for system design. Breaking down the system into classes with specific properties and methods allows writing a program with individual modules, which simplifies and clarifies programmer's work. Two formed basic matrices help obtain an object of the graph. The vertex matrix has number column and two x and y coordinates columns. The edges matrix first column presents edges start point numbers and the second edges end point numbers, the third column presents lengths of edges. Floyd-Warshall algorithm, for finding shortest path from one graph node to another, presents two matrices. The path matrix indicates from which intermediate node from node to node is the shortest way. The length matrix presents minimum distance between concrete nodes. Using path matrix the shortest path is found from the start node to the end node of the graph. Analysis of presented graph shows that the shortest path is very sensitive to even small coordinate changes of nodes. It shows that such systems are very important for optimal control of transport and other flows. A graphical environment and a working programming language in this environment are required for design of such systems. For example, Visual Basic for Application programming language works with the AutoCAD environment.

7. References

Autodesk (2001). *AutoCAD 2002. DXF Reference Guide*. Autodesk, Inc., 188 p.

Bajo, J.; Corchado, J.M.; De Paz, Y.; De Paz, J.F.; Rodrıguez, S.; Martın, Q.; & Abraham, A. (2009). SHOMAS: Intelligent guidance and suggestions in shopping centres, *Applied Soft Computing*, Vol. 9, pp. 851–862.

Chaib-draa, B.; Moulin, B.; Mandiau, R. & Millot, P. (1992). Trends in Distributed Artificial Intelligence, *Artificial Intelligence Review* Vol. 6, pp. 35-66.

Corchado, J.M.; Bajo, J.; de Paz, Y. & Tapia, D.I. (2008). Intelligent environment for monitoring Alzheimer patients, agent technology for health care, *Decision Support Systems* Vol. 44, pp. 382–396.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L. & Stein, C. (2001). *Introduction to Algorithms,* The MIT Press, New York.

Cottingham, M. (2001). *Mastering AutoCAD VBA*, Sybex, 656 p.

Diestel, R. (2000). *Graph theory,* Electronic edition. Springer – Verlag New York.

Doran, J.; Carvajal, H.; Choo, Y; & Li, Y. (1991). The MCS Multi-agent Testbed: Developments and Experiments, *Cooperating Knowledge based Systems*, Heidelberg: Springer-Verlag, pp. 240-251.

Dunn-Davies, H. R.; Cunningham, R. J.; & Paurobally, S. (2005). Propositional Statecharts for Agent Interaction Protocols, Electronic Notes in Theoretical Computer Science, Vol. 134, pp. 55–75.

Gasser, L. (1991). Social Conceptions of Knowledge and Action: DAI Foundations and Open Systems, *Artificial Intelligence* Vol. 47, pp. 107-138.

Gasser, L. , Rosenschein, J. S. & Ephrati, E. (1995). Introduction to Multi-Agent Systems, *Tutorial A Presented at the 1st International Conference on Multi-Agent Systems*, San Francisco, CA, June.

Haynes, S. R.; Cohen, M. A.; & Ritter, F. E. (2009). Designs for Explaining Intelligent Agents, *International Journal Human-Computer Studies*, Vol. 67, pp. 90–110.

Hewitt, C. (1977). Viewing Control Structures as Patterns of Passing Messages, *Artificial Intelligence* Vol. 8, No. 3, pp. 323-364.

Nwana H. S. (1996). Software Agents: an Overview, *Knowledge Engineering Review,* Vol. 11, No. 3, pp. 205-244.

Odell, J.; Parunak, H. V.-D.; & Bauer, B. (2000). Extending UML for agents. In: *Agent-Oriented Information Systems Workshop*, 17th National Conference on Artificial Intelligence, Austin, TX, USA, pp. 3–17.

Pan, A.; Leung, S.Y.S.; Moon, K.L.; & Yeung, K.W. (2009). Optimal reorder decision-making in the agent-based apparel supply chain, Expert Systems with Applications, Vol. 36, pp. 8571–8581.

Russel, S.J. & Norvig, P. (2009). Artificial Intelligence: A Modern Approach, Prentice-Hall, third edition.

Rumbaugh, J.; Jacobson, I.; & Booch, G. (1999). *The Unified Modeling Language Reference Manual*, Addison Wesley.

Sokas, A. (2005). Data exchange technology and database in engineering drawings. *Proceedings of the Third International Conference on Construction in the 21 st Century*. Advancing Engineering, Management and Technology. September 15-17, Athens, Greece, pp. 764-769.

Sokas, A. (2010). Intelligent agent find its way in the drawing. *Solid State Phenomena: Mechatronic Systems and Materials*: a collection of papers from the 5th international conference (MSM 2009), Vilnius, Lithuania, 23-25 October 2009. Uetikon-Zurich: Trans Tech Publications Inc. Vol. 165, pp. 425-430.

Sutphin, J. (2006). *AutoCAD 2006 VBA: Programmer's Reference*. Apress, 777 p.

Vallejo, D.; Albusac, J.; Castro-Schez, J.J.; Glez-Morcillo, C. ; & Jimenez, L. (2011). A multi-agent architecture for supporting distributed normality-based intelligent surveillance, *Engineering Applications of Artificial Intelligence*, Vol. 24, pp. 325–340.

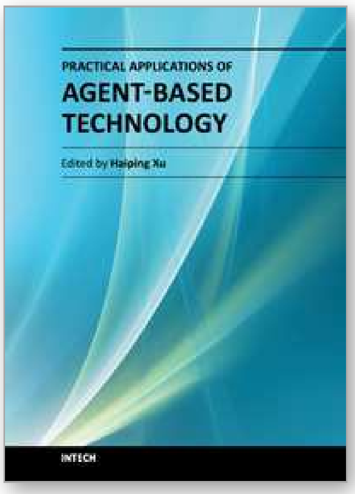
Wooldridge, M. (1995). Conceptualising and Developing Agents, In *Proceedings of the UNICOM Seminar on Agent Software,* 25-26 April, London, pp. 40-54.

Wooldridge, M. & Jennings, N. (1995a), "Intelligent Agents: Theory and Practice", *The Knowledge Engineering Review* Vol. 10, No. 2, pp. 115-152.

Wooldridge, M. & Jennings, N. (eds.) (1995b), Intelligent Agents, *Lecture Notes in Artificial Intelligence* Vol. 890, Heidelberg: Springer Verlag.

Wooldridge, M., Mueller, J. P. & Tambe, M. (1996), Intelligent Agents II, *Lecture Notes in Artificial Intelligence* Vol. 1037, Heidelberg: Springer Verlag.

Xiao, L. (2009). An adaptive security model using agent-oriented MDA. *Information and Software Technology*, Vol. 51, pp. 933–955.



Practical Applications of Agent-Based Technology

Edited by Dr. Haiping Xu

ISBN 978-953-51-0276-2
Hard cover, 136 pages
Publisher InTech
Published online 18, March, 2012
Published in print edition March, 2012

Agent-based technology provides a new computing paradigm, where intelligent agents can be used to perform tasks such as sensing, planning, scheduling, reasoning and decision-making. In an agent-based system, software agents with sufficient intelligence and autonomy can either work independently or coordinately with other agents to accomplish tasks and missions. In this book, we provide up-to-date practical applications of agent-based technology in various fields, such as electronic commerce, grid computing, and adaptive virtual environment. The selected applications are invaluable for researchers and practitioners to understand the practical usage of agent-based technology, and also to apply agent-based technology innovatively in different areas.