INTERNATIONAL JOURNAL OF RESEARCH IN COMPUTER APPLICATIONS AND ROBOTICS
ISSN 2320-7345

DESIGN & DEVELOPMENT OF A CLOUD COMPUTING ARCHITECTURE FOR AGENT-BASED URBAN TRANSPORTATION SYSTEMS

[1]Ms.Syeda Ayesha Thainiath, [2]Prof.Dr.G.Manoj Someswar

[1]Assistant Professor in Computer Science & Engineering at Nawab Shah Alam Khan College of Engineering & Technology(Affiliated to JNTUH), Malakpet, Hyderabad-500024, A.P. India.

[2]Professor, HOD & DEAN(Research), Department of Computer Science & Engineering at Nawab Shah Alam Khan College of Engineering & Technology (Affiliated to JNTUH), Malakpet, Hyderabad-500024, A.P. India.

[1] ashtahni@gmail.com, [2] manojgelli@gmail.com

Abstract: Agent-based traffic management systems can use the autonomy, mobility, and adaptability of mobile agents to deal with dynamic traffic environments. Cloud computing can help such systems cope with the large amounts of storage and computing resources required to use traffic strategy agents and mass transport data effectively. This article reviews the history of the development of traffic control and management systems within the evolving computing paradigm and shows the state of traffic control and management systems based on mobile multi agent technology. Intelligent transportation clouds could provide services such as decision support, a standard development environment for traffic management strategies, and so on. With mobile agent technology, an urban-traffic management system based on Agent-Based Distributed and Adaptive Platforms for Transportation Systems (Adapts) is both feasible and effective. However, the large-scale use of mobile agents will lead to the emergence of a complex, powerful organization layer that requires enormous computing and power resources.

To deal with this problem, we propose a prototype urban-traffic management system using intelligent traffic clouds.

Local area networks (LANs) appeared to enable resource sharing and handle the increasingly complex requirements. One such LAN, the Ethernet, was invented in 1973 and has been widely used since. During the same period, urban-traffic-management systems took advantage of LAN technology to develop into a hierarchical model. Network communication enabled the layers to handle their own duties while cooperating with one another.

In the following Internet era, users have been able to retrieve data from remote sites and process them locally, but this wasted a lot of precious network bandwidth. Agent based computing and mobile field. From multi agent systems and agent structure to ways of negotiating between agents to control agent strategies, all these fields have had varying degrees of success [2,3]. Now, the IT industry has ushered in the fifth computing paradigm: cloud computing. Based on the Internet, cloud computing provides on demand computing capacity to individuals and businesses in the form of heterogeneous and autonomous services. With cloud computing, users do not need to understand the details of the infrastructure in the “clouds;” they need only know what resources they need and how to obtain appropriate services, which shields the computational complexity of providing the required services.

In recent years, the research and application of parallel transportation management systems (PtMS), which consists of artificial systems, computational experiments, and parallel execution, has become a hot spot in the traffic research field. Here, the term parallel describes the parallel interaction between an actual transportation system and one or more of its corresponding artificial or virtual counterparts.[4] Such complex systems make it difficult or even impossible to build accurate models and perform experiments, so PtMSs use artificial transportation systems (ATS) to compensate for this defect. Moreover, ATSs also help optimize and evaluate large amounts of traffic-control strategies. Cloud computing caters to the idea of "local simple, remote complex" in parallel traffic systems. Such systems can take advantage of cloud computing to organize computing experiments, test the performance of different traffic strategies, and so on. Thus, only the optimum traffic strategies will be used in urban-traffic control and management systems.

.

Keywords: Artificial Transportation systems, Adaptive Platforms for Transportation Systems, mobile agent technology, Agent monitor testing, intelligent traffic clouds, parallel transportation management systems.

INTRODUCTION

Agent-based traffic management systems can use the autonomy, mobility, and adaptability of mobile agents to deal with dynamic traffic environments. Cloud computing can help such systems cope with the large amounts of storage and computing resources required to use traffic strategy agents and mass transport data effectively. This research paper reviews the history of the development of traffic control and management systems within the evolving computing paradigm and shows the state of traffic control and management systems based on mobile multi agent technology. Intelligent transportation clouds could provide services such as decision support, a standard development environment for traffic management strategies, and so on. With mobile agent technology, an urban-traffic management system based on Agent-Based Distributed and Adaptive Platforms for Transportation Systems (Adapts) is both feasible and effective. However, the large-scale use of mobile agents will lead to the emergence of a complex, powerful organization layer that requires enormous computing and power resources.

To deal with this problem, we propose a prototype urban-traffic management system using intelligent traffic clouds.

OBJECTIVES

Agent-based computing and mobile agents were proposed to handle this vexing problem. Only requiring a runtime environment, mobile agents can run computations near data to improve performance by reducing communication time and costs. This computing paradigm soon drew much attention in the transportation field. From multi agent systems and agent structure to ways of negotiating between agents to control agent strategies, all these fields have had varying degrees of success.

SYSTEM DESIGN

ARCHITECTURE DIAGRAM

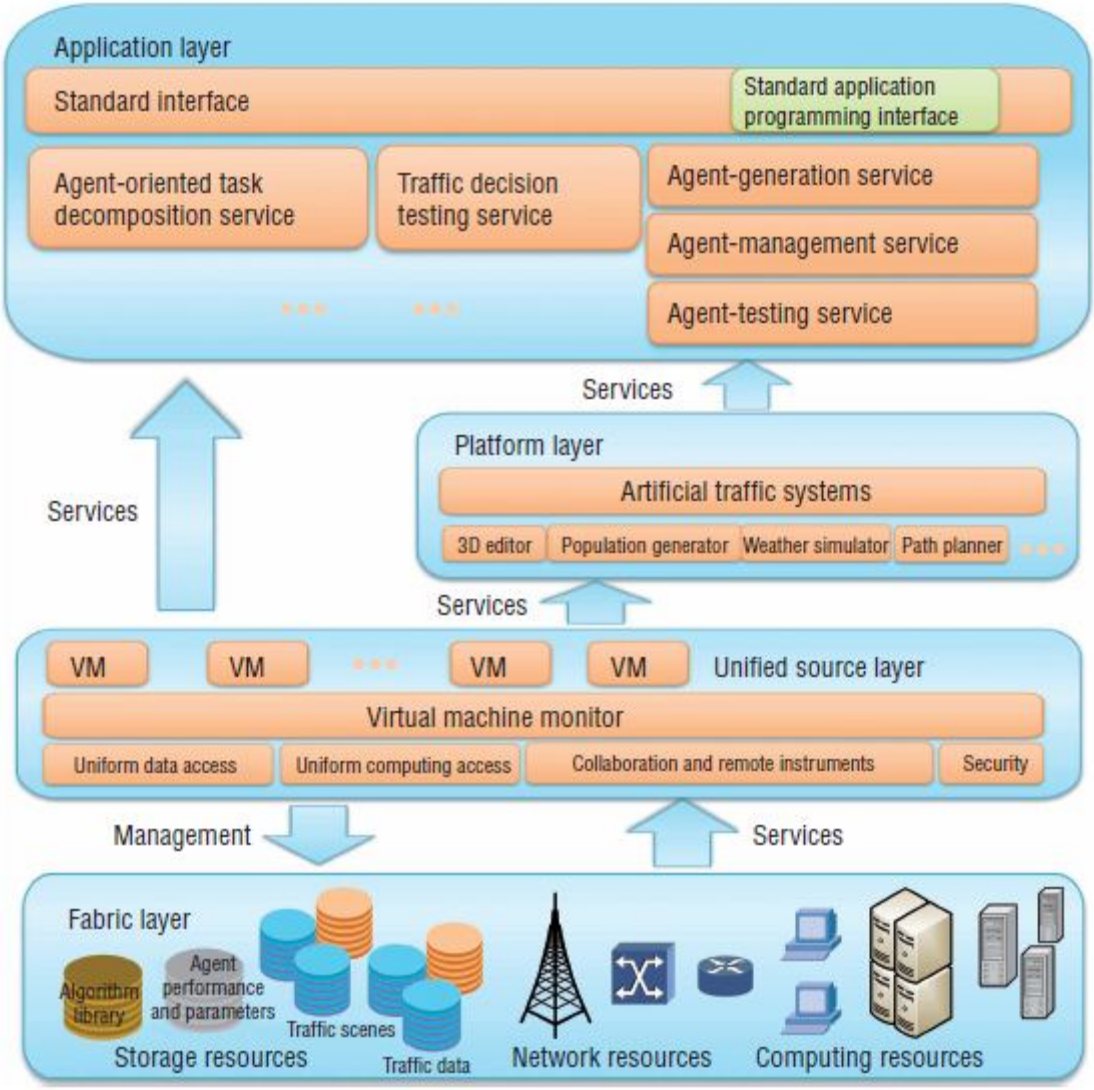


Figure 1: ARCHITECTURE DIAGRAM

BLOCK DIAGRAM

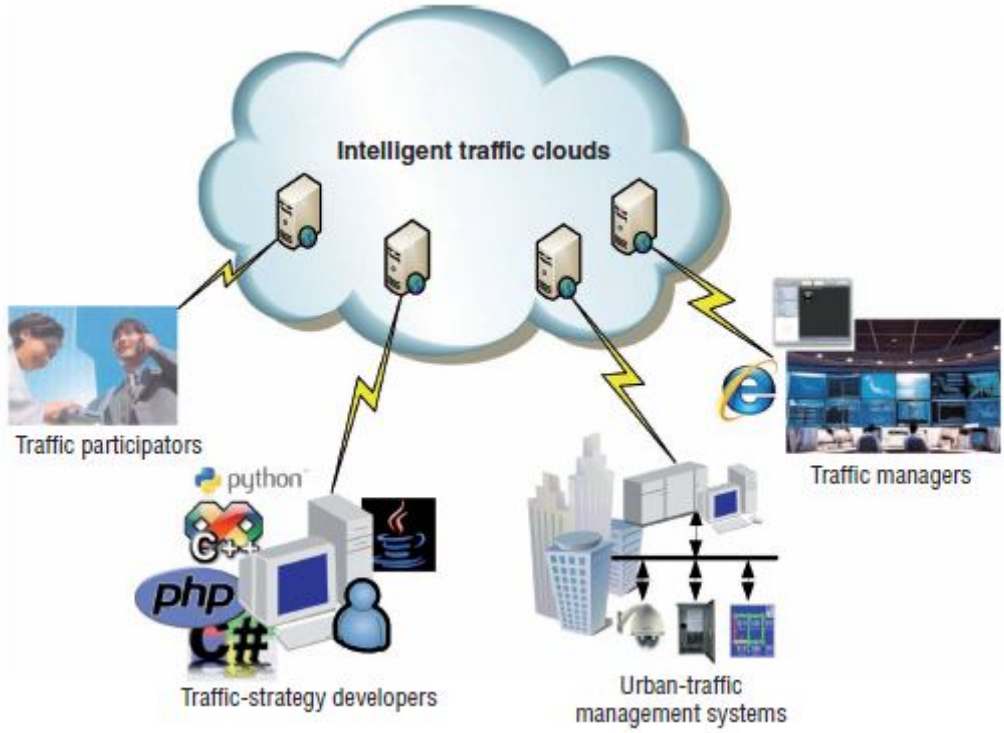
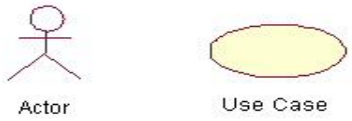


Figure 2: BLOCK DIAGRAM

UML DIAGRAMS

USECASE DIAGRAM

A use case is a set of scenarios that describing an interaction between a user and a system. A use case diagram displays the relationship among actors and use cases. The two main components of a use case diagram are use cases and actors.



An actor is represents a user or another system that will interact with the system you are modeling. A use case is an external view of the system that represents some action the user might perform in order to complete a task.

Use cases are used in almost every project. They are helpful in exposing requirements and planning the project. During the initial stage of a project most use cases should be defined, but as the project continues more might become visible.

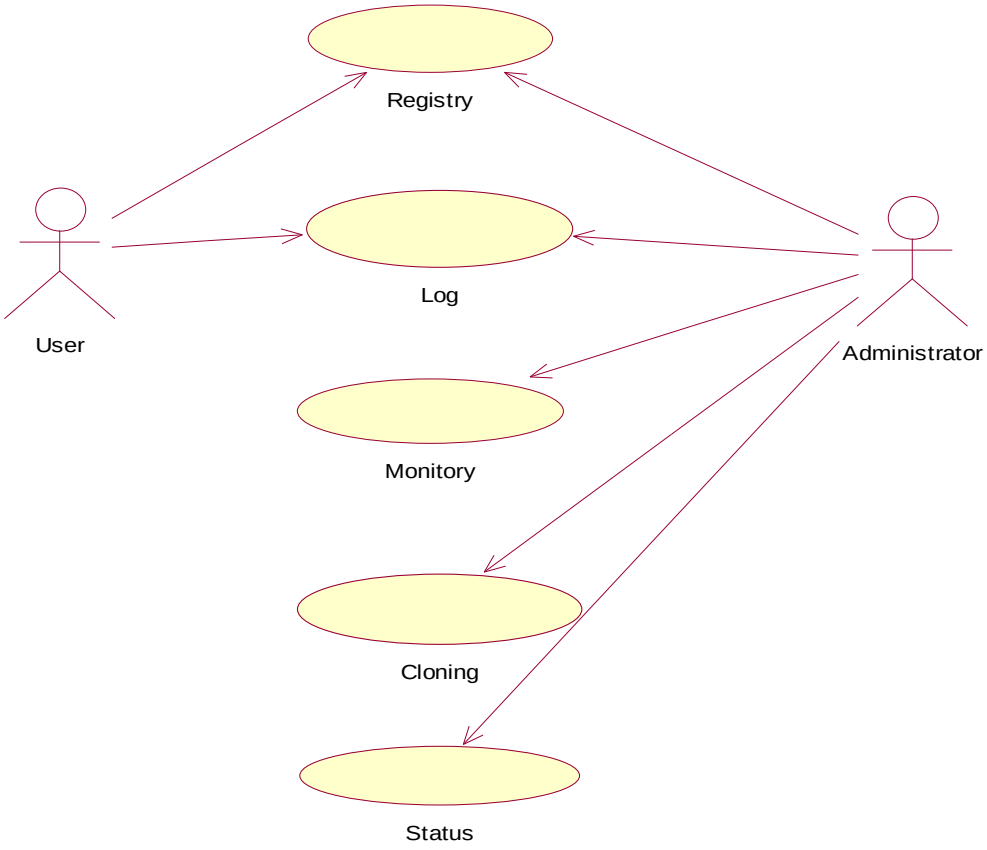


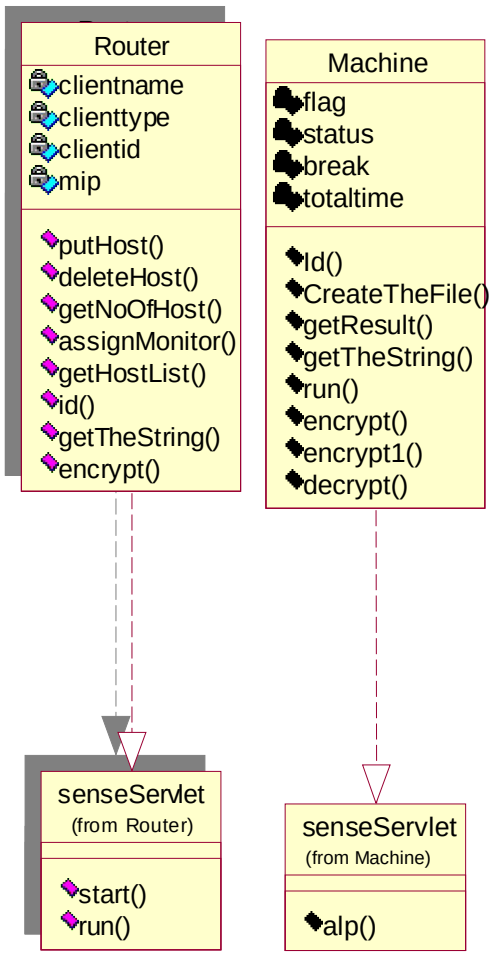
Figure 3: USE CASE DIAGRAM

In this diagram, the left hand side actor represents the user that is which defines the action of the user. The user need to register with the RMI then only they are able to access the process. After registry, they need to login to the process then the user able to access the process. With the help of administrator, the central machine able to monitor all the machines involved in that network. Next they concentrate on the cloning process, that is the mobile agents need to migrate the control from one node to another. So, with the help of this diagram the other people able to understand the process which are doing in the project.

Class diagram:

Class diagrams are widely used to describe the types of objects in a system and their relationships. Class diagrams model class structure and contents using design elements such as classes, packages and objects. Class diagrams describe three different perspectives when designing a system, conceptual, specification, and implementation. These perspectives become evident as the diagram is created and help solidify the design. This example is only meant as an introduction to the UML and class diagrams. If you would like to learn more see the Resources page for more detailed resources on UML.

Classes are composed of three things: a name, attributes, and operations. Below is an example of a class.



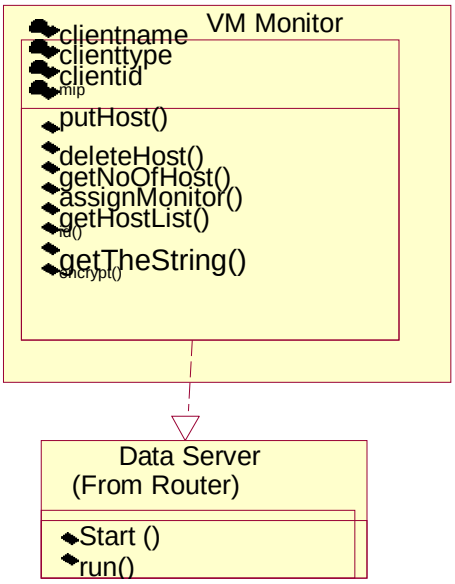


Figure 4: CLASS DIAGRAM

With the help of the class diagram we are able to find the attributes and methods involved in the class. In this agent service class diagram they have the attributes like client id, client name, client type and machine ip .All the attributes which are used in this class methods and they return some values for each function. The client id must be used for finding the host name which are used to find which machine are currently in the network. Start and run are the methods which are used for starting and running the threads.

In Router class, it contains the attributes like clienttype, clientid, mip which are used to run the methods.In the third part represents the methods which are involved in that class. The thread called "senseservlet" which are involved in this class. In senseservlet thread which contains the methods called start() and run().The start method starts the execution of the invoking object. It can throw an IllegalThreadStateException if the thread was already started.The run method runs the thread program which contains the set of instructions.The run method is automatically called after the start method. A thread is said to be in runnable state, when it is executing a set of instructions.

In machine class diagram, flag, status, break and total time are the attributes which are used to find the total time required for the program execution. A thread sense control is a line of execution which contains the small unit of code that is dispatched with the scheduler. The main thread can continue with the execution of the rest of the program. The thread having the method called "alp" which are used to encrypt and decrypt the information at the program execution. In this class, we have some methods like id() which are used to get the id of the client id similarly all the methods will perform respectively.

Sequence diagram:

UML sequence diagrams are used to represent or model the flow of messages, events and actions between the objects or components of a system. Time is represented in the vertical direction showing the sequence of interactions of the header elements, which are displayed horizontally at the top of the diagram.

Sequence Diagrams are used primarily to design, document and validate the architecture, interfaces and logic of the system by describing the sequence of actions that need to be performed to complete a task or scenario. UML sequence diagrams are useful design tools because they provide a dynamic view of the system behavior which can be difficult to extract from static diagrams or specifications.

Although UML sequence diagrams are typically used to describe object-oriented software systems, they are also extremely useful as system engineering tools to design system architectures, in business process engineering as process flow diagrams, as message sequence charts and call flows for telecom/wireless system design, and for protocol stack design and analysis.

Sequence Diagram Header Element

Sequence diagrams are particularly useful for modeling:

- **Complex interactions between components.** Sequence diagrams are often used to design the interactions between components of a system that need to work together to accomplish a task. They are particularly useful when the components are being developed in parallel by different teams (typical in wireless and telephony systems) because they support the design of robust interfaces that cover multiple scenarios and special cases.[7]
- **Use case elaboration.** Usage scenarios describe a way the system may be used by its actors. The UML sequence diagram can be used to flesh out the details of one or more use cases by illustrating visually how the system will behave in a particular scenario. The use cases along with their corresponding sequence diagrams describe the expected behavior of the system and form a strong foundation for the development of system architectures with robust interfaces.[9]
- **Distributed & web-based systems.** When a system consists of distributed components (such as a client communicating with one or more servers over the Internet), sequence diagrams can be used to document and validate the architecture, interfaces and logic of each of these components for a set of usage scenarios.[6]
- **Complex logic.** UML sequence diagrams are often used to model the logic of a complex feature by showing the interactions between the various objects that collaborate to implement each scenario. Modeling multiple scenarios showing different aspects of the feature helps developers take into account special cases during implementation.[5]
- **State machines.** Telecom, wireless and embedded systems make extensive use of state machine based designs where one or more state machines communicate with each other and with external entities to perform their work. For example, each task in the protocol stack of a cellular phone goes through a series of states to perform actions such as setup a call or register with a new base station. Similarly the call processing components of a Mobile Switching Center use state machines to control the registration and transfer of calls to roaming subscribers. Sequence diagrams (or call flows as they are commonly referred to in the telecom and wireless industry) are useful for these types of applications because they can visually depict the messages being exchanged between the components and their associated state transitions.[12]

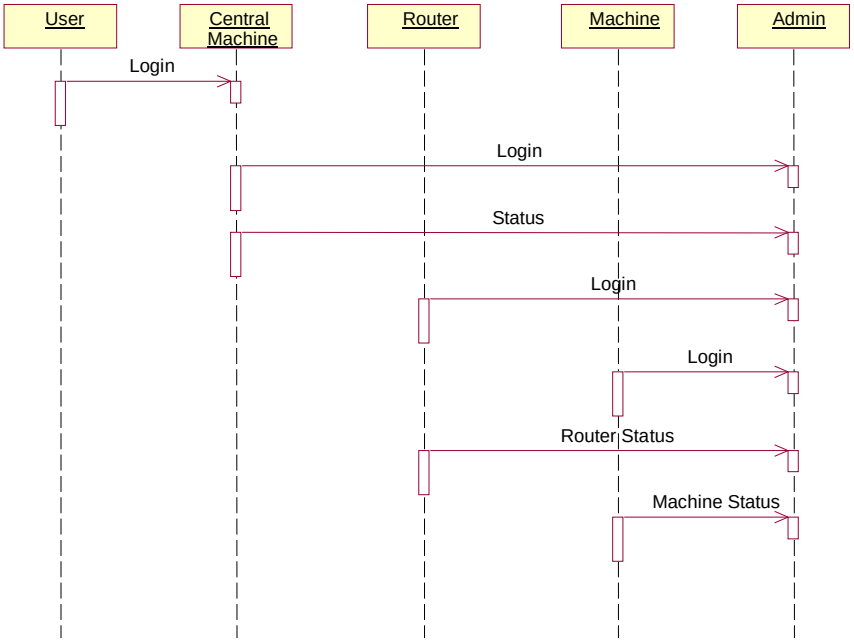


Figure 5: LOGIN SEQUENCE DIAGRAM

In the above sequence diagram which represents the sequence of actions that are made at the time of login. That is, the user login to the process, then the user connected with the central machine then all the controls are login into the process they are Router, Machine, etc., so all are login to the process then each and every machine status are monitored by the central machine. The machines are connected with the router then they are monitored by their area monitors. Then all the routers are monitored by the central machine.

DEVELOPMENT

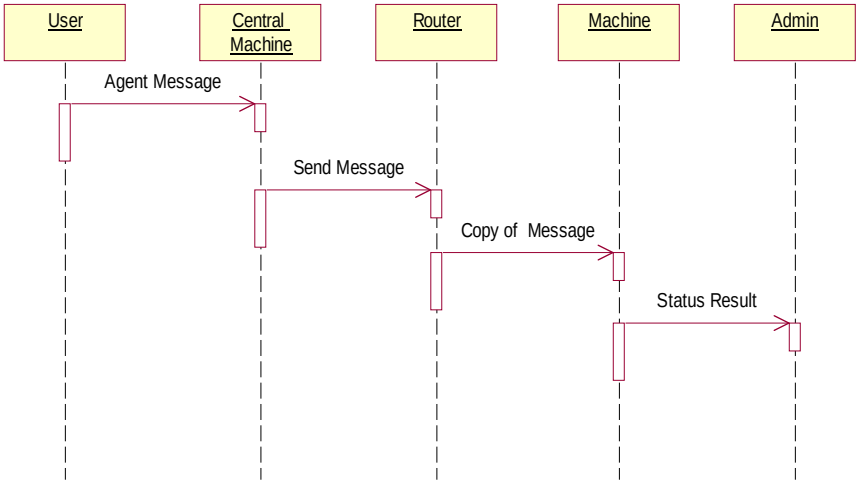


Figure 6: DEVELOPMENT SEQUENCE DIAGRAM

In this sequence diagram, which explains the sequence of actions that are happened at the time of development process? In this process, if the user sends any process to the central machine then it sends the message to the router. Then the router sends the copy of that message to the machines involved in that network. Finally, the administrator got the result from the machine and it sends the status to the central machine. So, the central machine able to get the status of each and every machine with the help of their area monitors.

LOGOFF

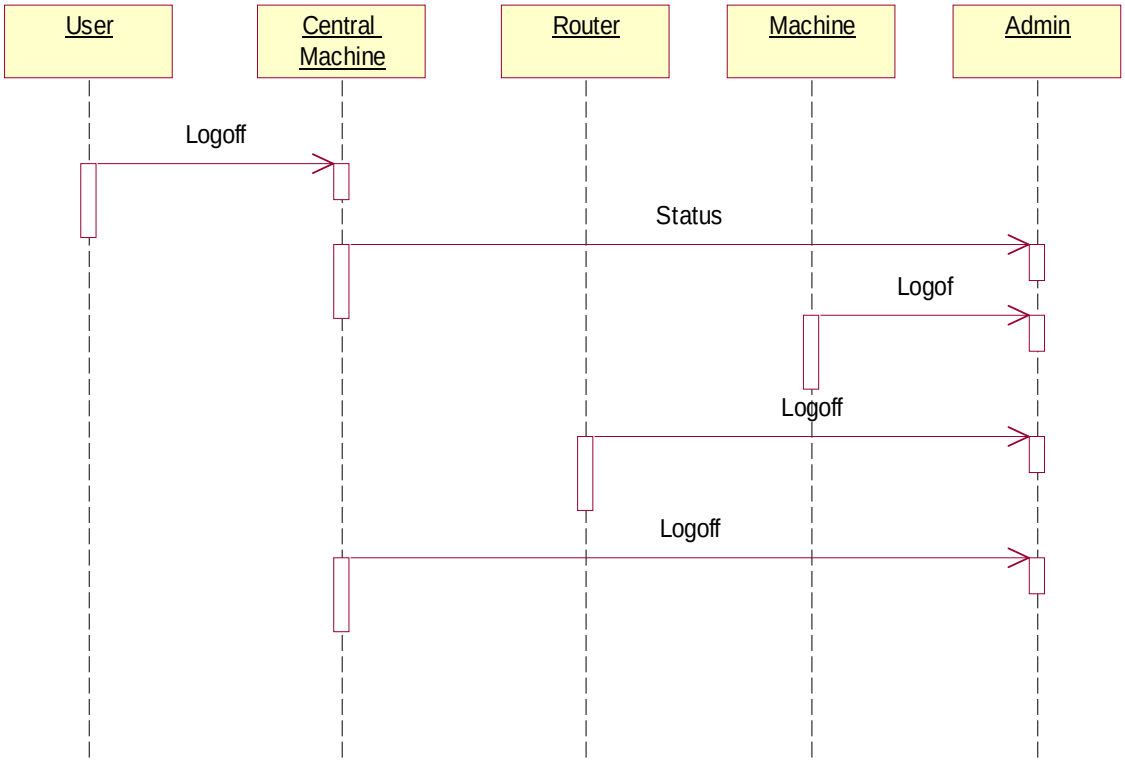


Figure 7: LOGOFF SEQUENCE DIAGRAM

In logoff process sequence diagram, which explaining the sequence of steps are followed at the time of logoff process. So, the user need to logoff the machine then the status of that machine will send it to the central machine. If the machines are logoff then automatically the router of that machine will logoff. So each every action of the process that are monitored by the central machine. The central machine not able to monitor all the machines in the network so for that purpose they are using routers. If the machine is added to the network then they are calculating the heuristic function to find out the threshold value.

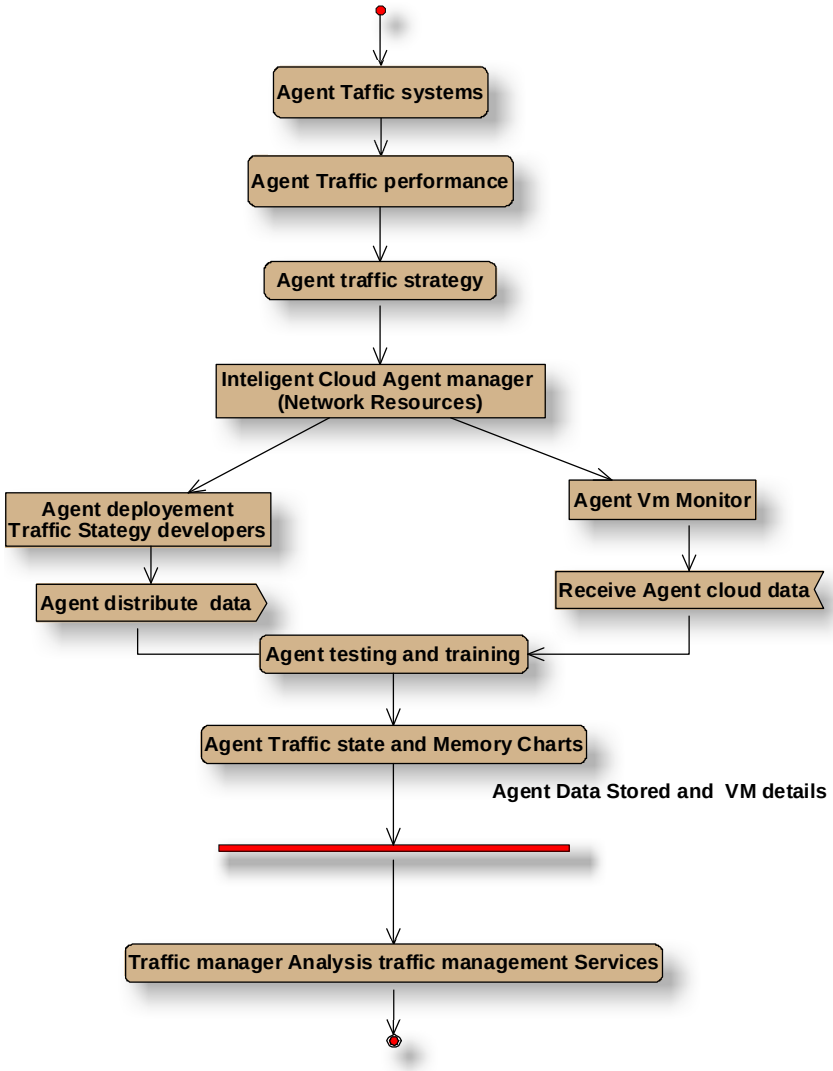


Figure 8: ACTIVITY DIAGRAM

TESTING

TESTING METHODOLOGY:

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

UNIT TESTING:

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program input produces valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests

perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.[11]

INTEGRATION TESTING:

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Integration testing is of three types:

- Bottom up Integration
- Top down Integration
- Sandwich Integration

Bottom up integration testing consists of unit testing followed by system testing. Unit testing has the goal of testing individual modules in the system. Subsystem testing is concerned with verifying the operation of the interfaces between modules in the sub systems. Top down integration testing starts with the main routine and one or two immediately subordinate routines in the system structure. Top down integration requires the use of program stubs to simulate the effect of lower level routines that are called by those being tested.[8]

TOP DOWN METHOD HAS THE FOLLOWING ADVANTAGES:

- System integration is distributed through the implementation phase. Modules are integrated as they are developed.
- Top level interfaces are tested first and most often.
- The top level routine provides a natural test harness for lower level routines.
- Errors are localized to the new modules and interfaces that are being added.
- **FUNCTIONAL TEST:**
- Functional tests provide a systematic demonstration that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals. Functional testing is cantered on the following items:
- Valid Input : Identified classes of valid input must be accepted.
- Invalid Input : Identified classes of invalid input must be rejected.
- Functions : Identified functions must be exercised.
- Output : Identified classes of application outputs must be exercised.
- Systems : Interfacing systems or procedures must be invoked.
- Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.[7]
- **SYSTEM TEST:**

- System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

- **WHITE BOX TESTING:**

- White Box Testing is a testing in which the software tester has knowledge of the inner working, structure and language of the software, or at least its purpose. It is used to test areas that cannot be reached from a black box level. [1]

- **BLACK BOX TESTING:**

- Black Box Testing is a testing in which the software tester test the system without any knowledge of the inner working structure or language of the module being tested. It is a testing in which the software under test is treated, as a black box, you cannot "see" into it. The test provides inputs and responds to outputs without considering how the software works. [4]

- **TESTING UNDER VARIOUS STAGES:**

- **UNIT TESTING:**

- Unit testing is usually conducted as part of a combined code and unit test phase of the software life cycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

TESTING STRATEGY AND APPROACH:

Field testing will be performed manually and functional tests will be written in detailed.

TEST OBJECTIVES:

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

FEATURES TO BE TESTED:

- Verify that the entries are of the correct format..
- No duplicate entries should be allowed.
- All links should take the user to the correct page.

INTEGRATION TESTING:

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.[5]

TEST RESULTS: All the test cases mentioned below passed successfully. No defects encountered.

ACCEPTANCE TESTING:

User Acceptance Testing is a critical phase pf any project and requires significant participation by end user. It also ensures that the system meets the functional requirements.[6]

TEST RESULT: All the test cases mentioned below passed successfully. No defects encountered.

TEST CASES:

TEST CASE FOR REGISTERY

Test Case Name	Register rmi
Purpose:	To ensure that rmi is register
Input:	Registry
Expected Result:	Connect to agent monitor
Actual Result:	Connection refused
Failure	rmi registry is not invoked
Remarks	Test Case Passed

TABLE 1: REGISTRY

TEST CASE FOR AGENT MONITOR:

Test Case Name	To test Run Agent Monitor
Purpose:	To ensure that run agent monitor is running
Input:	Agent services on
Expected Result:	GUI
Actual Result:	Blank screen
Failure	Java compilation is required
Remarks	Test Case Passed

TABLE 2: AGENT MONITOR TESTING

TEST CASE FOR ROUTER:

Test Case Name	Routers
Purpose:	To ensure that routers should be registered
Input:	Routers
Expected Result:	3 routers has been activated
Actual Result:	3 routers should be activated
Failure	nil
Remarks	Test Case Passed

TABLE 3: Routers

TEST CASE FOR SUCCESSFUL TASK COMPLETION :

Test Case Name	To test the tasks completed successfully
Purpose:	To ensure all the individual tasks completed successfully
Input:	The task assigned with the proper resources needed
Expected Result:	The tasks had to be completed with proper outputs
Actual Result:	The tasks completed with the proper expected outputs
Failure	Improper Task scheduling /Processing/ coding errors
Remarks	Test Case Passed

TABLE 4: TASK COMPLETION TESTING

Existing system:

Software agents and their applications in traffic and transportation systems have been studied for over one decade. A number of agent-based applications have already been reported in the literature. These applications propose and investigate different agent-based approaches in various traffic and transportation related areas. The research results

clearly demonstrate the potential of using agent technology to improve the performance of traffic and transportation systems. Most agent based applications, however, focus on modelling and simulation.

Few real-world applications are implemented and deployed. In general, the design, implementation, and application of agent based approaches in the area of traffic and transportation are still immature and need to be further studied. The integration of new technologies, such as mobile agent technology, should be considered to enhance the flexibility of systems and the ability to deal with uncertainty in dynamic environments.

Monitoring:

Virtualization brings to Clouds is the potential difficulty in fine-control over the monitoring of resources. Although many Grids (such as TeraGrid) also enforce restrictions on what kind of sensors or long-running services a user can launch, Cloud monitoring is not as straightforward as in Grids, because Grids in general have a different trust model in which users via their identity delegation can access and browse resources at different Grid sites, and Grid resources are not highly abstracted and virtualized as in Clouds.

Provenance:

Provenance refers to the derivation history of a data product, including all the data sources, intermediate data products, and the procedures that were applied to produce the data product. Provenance information is vital in understanding, discovering, validating, and sharing a certain data product as well as the applications and programs used to derive it.

Security Model:

Clouds mostly comprise dedicated data centres belonging to the same organization, and within each data centre, hardware and software configurations, and supporting platforms are in general more homogeneous as compared with those in Grid environments. Interoperability can become a serious issue for cross-data centre, cross-administration domain interactions, imagine running your accounting service in Amazon EC2 while your other business operations on Google infrastructure. Grids however build on the assumption that resources are heterogeneous and dynamic, and each Grid site may have its own administration domain and operation autonomy.

Cloud infrastructure typically rely on Web forms (over SSL) to create and manage account information for end-users, and allows users to reset their passwords and receive new passwords via Emails in an unsafe and unencrypted communication. Note that new users could use Clouds relatively easily and almost instantly, with a credit card and/or email address.

CONCLUSION AND FUTURE WORK

Urban traffic management systems based on cloud computing. The intelligent traffic clouds could provide traffic strategy agents and agent-distribution maps to the traffic management systems, traffic-strategy performance to the traffic-strategy developer, and the state of urban traffic transportation and the effect of traffic decisions to the traffic managers. It could also deal with different customers' requests for services such as storage service for traffic data and strategies, mobile traffic-strategy agents, and so on.

With the development of intelligent traffic clouds, numerous traffic management systems could connect and share the clouds' infinite capability, thus saving resources, time and costs.

REFERENCES

1. D.C. Gazis, "Traffic Control: From Hand Signals to Computers," Proc. IEEE, vol. 59, no. 7, 1971, pp. 1090–1099.

2. F.-Y. Wang, "Parallel System Methods for Management and Control of Complex Systems," Control and Decision, vol. 19, no. 5, 2004, pp. 485–489.

3. F.-Y. Wang, "Toward a Revolution in Transportation Operations: AI for Complex Systems," IEEE Intelligent Systems, vol. 23, no. 6, 2008, pp. 8–13.

4. F.-Y. Wang, "Parallel Control and Management for Intelligent Transportation Systems: Concepts, Architectures, and Applications," IEEE Trans. Intelligent Transportation Systems, vol. 11, no. 3, 2010, pp. 1–9.

5. B. Chen and H. H. Cheng, "A Review of the Applications of Agent Technology in Traffic and Transportation Systems," IEEE Trans. Intelligent Transportation Systems, vol. 11, no. 2, 2010, pp. 485–497.

6. M.C.Choy, D.Srinivasan and R.L.Cheu, "Cooperative, Hybrid Agent Architecture for Real-Time Traffic Signal Control," IEEE Trans. Systems, Man and Cybernetics, Part A: Systems and Humans, vol. 33, no. 5, 2003, pp. 597–607.

7. M.C.Choy, D.Srinivasan, and R.L.Cheu, "Neural Networks for Continuous Online Learning and Control," IEEE Trans. Neural Networks, vo1. 7, no. 6, 2006, pp. 1511–1531.

8. B.P. Gokulan and D. Srinivasan, "Distributed Geometric Fuzzy Multiagent Urban Traffic Signal Control," IEEE Trans. Intelligent Transportation Systems, vol. 11, no. 3, 2010, pp. 714–727.

9. N. Suri, K.M. Ford, and A.J. Cafias, "An Architecture for Smart Internet Agents," Proc. 11th Int'l FLAIRS Conf., AAAI Press, 1998, pp. 116–120.

10. F.-Y. Wang, "Agent-Based Control for Networked Traffic Management Systems," IEEE Intelligent Systems, vol. 20, no. 5, 2005, pp. 92–96.

11. F.-Y. Wang and S. Tang, "Artificial Societies for Integrated and Sustainable Development of Metropolitan Systems," IEEE Intelligent Systems, vol. 19, no. 4, 2004, pp. 82–87.

12. I. Foster et al., "Cloud Computing and Grid Computing 360-Degree Compared," Proc. Grid Computing Environments Workshop, IEEE Press, 2008, pp. 1–10.