Adaptable Fault-tolerant Performance Monitoring System based on Autonomous Mobile Agents

Vasileios A. Baousis

Submitted for the Degree of
Master of Science in Pathway
Mobile and Satellite Communications
from the
University of Surrey

Department of Electronic Engineering
School of Electronics and Physical Sciences
University of Surrey
Guildford, Surrey, GU2 7XH, UK

August 2003

Supervised by: Professor George Pavlou

Table of Context

1 Introduction 1
1.1 Overview 1
1.2 Problem definition and Objectives 1
2 Background 3
2.1 Software Agents 3
2.1.1 Mobile Agent Systems 4
2.2 Network Performance Monitoring 8
2.2.1 Active and Passive Measurements 8
2.2.2 Uniformly Weighted Moving Average (UWME) 10
2.2.3 Severity indicating gauge-threshold model 10
3 Work done 12
3.1.1 Requirements 12
3.1.2 Specifications 14
3.1.3 Design 21
4 Implementation 38
5 Achievements 49
6 Conclusions and Future Directions 50
7 References 51
8 Appendix: Source code. 54

List of figures

Figure 1 The severity gauge threshold model (with one threshold) 11
Figure 2 Worker's sub modules and their interfaces 18
Figure 3 Interfaces class diagrams 22
Figure 4 Project's objects 23
Figure 5 Project's data types 23
Figure 6 Proximity failure sequence diagram 29
Figure 7 Correspondent unavailable sequence diagram 32
Figure 8 Collaborator unavailable sequence diagram 33
Figure 9 Resource unavailable sequence diagram 34
Figure 10 Logic unavailable sequence diagram 36
Figure 11 Unavailability of operational resources 37
Figure 12 System's setup 39
Figure 13 Performance Monitor Graphical User Interface. 39
Figure 14 System's registry 40
Figure 15 Location agent collects inputs from the first host (169.254.212.10) 41
Figure 16 Location agent collects inputs from the second host (169.254.154.26) 41
Figure 17 LocationAgent creation and removal after its task completion 42
Figure 18 GUI for user interaction with the IAEM. 43
Figure 19 System's properties GUI. 44
Figure 20 Master's Agent creation (Correspondent unavailable failure) 45
Figure 21 Resource unavailable failure. 45
Figure 22 Collaborator unavailable failure 46
Figure 23 Logic unavailable failure handling 47
Figure 24 Migration denied failure 48
Figure 25 System's API 49

To my beloved

wife Helena

and

little daughter Alexandra

for their support, endurance and encouragement

throughout this very demanding year.

1 Introduction

1.1 Overview

This project concentrates on the enhancement of a network performance monitoring system with the autonomous behaviour needed in order to proactively or reactively deal with critical conditions. "Intelligent" software entities (stationary or mobile agents) sense the changes in their environment, analyse them and execute the appropriate emergency procedures or actions that allow system operation even when a critical failure is encountered or imminent. The implementation of this project is based on the usage of the Mobile Agent technology (MAT).

1.2 Problem definition and Objectives

Modern data communication systems are naturally heterogeneous, composed of several and varied components, and yet must intercommunicate, interoperate, collaborate, share the same resources, and exchange data. In this environment some applications require a certain level of communication performance over the network that might be critical to their effectiveness. Performance management of a computer network is separated into two categories: **monitoring** and **controlling**. Monitoring involves the functions that keep track of network activities and controlling includes the functions that enable performance management so as making the appropriate network adjustments in order to improve or maintain some level of performance. Performance management system has to maintain knowledge of network resource status, by monitoring these resources and thus determining network operational level. Monitoring and controlling operations include:

- Monitoring network traffic.
- Usage of the network's traffic collected information, for network capacity planning and dimensioning.
- Monitoring of resource utilization by applying appropriate indicator levels and associated alarm notifications.
- Identification of network congestions and bottlenecks, and the appropriate recovery actions to be performed.
- Monitoring the QoS offered.

In this monitoring and controlling processes of a network performance monitoring system, several problems and failures might occur.

Another functional area of a network management system is the fault management. The objective of a fault management system is to identify faults and failures before or after their occurrence and this should take place as soon as possible. Additionally, it has also to identify the cause of the fault so that the appropriate system fault handling strategy to be followed and consequently, a remedial action to be taken [4]. These operations of the fault monitoring management system should include the following functions:

- Malfunctions report handling.
- Malfunction correlation, diagnostic and confidence testing.
- Fault identification and diagnosis.
- Dispatch of maintenance patches such as periodic testing and repair activities.
- Repair or modification of the failed components so as the network to be restored to its stable state.
- Provision of soft reconfiguration for fault bypassing.

The objectives of a fault tolerant system are more difficult to be met in a complex and diverse network.

The aim of this project is to enhance an already existing network performance monitoring system, based on mobile agents, with the intelligence, flexibility and autonomy needed, in order to handle several failures that might be encountered during the network performance motoring operation. The existing network performance monitoring system is enhanced with functionality similar to that of the fault management. The enhanced network performance monitoring system contains the intelligence that allows the autonomous fault management/handling of the performance monitoring components. Failures and errors are dealt by the new system, without user's interaction or involvement. The system is able to handle the following categories of failures:

- Communication failures.
- Availability of network performance monitoring entities.
- Required updates of management functionality.
- Resource over-utilization and agent migration issues.

2 Background

In this section the necessary background knowledge required to comprehend the various aspects of this project is provided. More specifically, it will be explained what a software and Mobile Agent is, and in particular their applicability into the framework of a network performance monitoring system.

2.1 Software Agents

There are many definitions about what a Software Agent (SA) is, but it would be considered as more appropriate to start with a general definition of the word **Agent** and then move to the description of the Software agent. According to the Oxford dictionary an "Agent is a person who acts for, or manages the affairs of, other people in business, politics etc". Similarly, to the above definition and moving to the software engineering word "A Software agent is a software entity that accomplishes some predefined operations on behalf of its owner, user or some other software entity which has tasked it. This Software agent performs its task with some degree of "independence or autonomy".

The above definition of software agents may add nothing to the ordinary perception of object-oriented software entities if it is regarded solely by this definition. Software agents differ from conventional (object-oriented) software technology in one or more of the following dimensions:

- **Autonomy**. Any agent has a measure of autonomy defined by its user. An autonomous agent can pursue agenda independently of its user. This requires aspects of periodic action, spontaneous execution and initiatives, so that the agent to be able to take pre-emptive or independent actions that would eventually benefit the user.
- **Proactiveness**. Pro-active agents generally follow plans, or at least execute rules when the environment reaches a known or predefined threshold. An agent recognizes the imminent condition and reacts in order to complete an operation or prevent a fault happening in the environment.
- **Distributedness**. Many different kinds of agents can work together in the same system, and be added or removed without interrupting it.
- **Planning**. The agent organizes the actions, which has to perform during its life by priorities. Planning is regarded as to be one of the most important properties for an intelligent agent to possess. Planning is used by deliberative and pro-active agents, according to their knowledge of their environment and in conjunction with the possible actions which agent can apply to it.

- **Delegation**. An agent may ask some other software entity to perform one of its goals or tasks. This capability is very important to achieve resource balancing, and distributed computation.
- **Co-operation**. The user and the agent are essentially collaborating in constructing a contract. The user is specifying what actions should be performed on his or her behalf and the agent is specifying what is capable of performing so as to provide the desired results. Co-operation may be achieved on a multilevel-multiparty fashion with the involvement of many users collaborating with many agents and also between two or more agents.
- **Learning**. An agent is able to change its behaviour based on its previous experience. For example, in a Network Management System (NMS) an agent should be able to acquire new capabilities in management, such as the ability to handle new types of network problems and faults.

2.1.1 Mobile Agent Systems

Mobility is an orthogonal property of agents—which means that not all agents are mobile. An agent can just sit in its environment of creation and communicate with its surroundings by conventional means, such as various forms of remote procedure calling and messaging. The agents that don't move are just called Stationary Agents [5].

A stationary agent runs only on the system where it begins execution. If it needs information that is not provided on that system or needs to interact with an agent on a different system, it typically uses a communication mechanism such as the Remote Procedure Call (RPC), Remote Method Invocation (RMI), etc.

On the other hand, a mobile agent is not bound to the system where it begins execution. The mobile agent is free to travel among the available network hosts. After its creation in one execution environment, it can transport its **execution state** and **code** with it to another environment in the network, where it resumes execution.

The term execution state typically means, the attribute values of the agent that help it determine what to do when it resumes execution at its destination (serialization-de-serialization processes take place in order for these attribute values to be transported across the network). The term code means, in an object-oriented content, the source code (classes) necessary for the agent to execute.

A mobile agent has the unique ability to transport itself from one system in a network to another. The ability to travel allows a mobile agent to move to a system that contains an object with which the agent wishes to interact and take advantage of being in the same host or network as the collaborating object.
In this project, two types of agent mobility are implemented: constrained and weak mobility. There is one additional type of agent mobility, strong mobility, but this is not examined nor used in this project.

2.1.1.1 Mobility Strategies

2.1.1.1.1 Constrained Mobility

Mobile agents can migrate from one node to another, perform their task, clone themselves, cooperate with other agents etc [8]. This type of behaviour is termed as “full mobility”. An alternative behaviour is a mobile agent to move from its node of creation to another node, guided probably by a “parent- administrative” stationary or mobile agent and stay to the last visited node until its task is performed. This behaviour is termed as “constrained mobility”. In this type of behaviour a mobile agent can be exploited in management environments. In constrained mobility the agent is created by a client, acting in the manager role, and then is dispatched to the target network element acting in the server role.
In constrained mobility, the agent does not need to have complex migration features and, as a result, its size and the incurred network traffic are minimal if compared to the other forms of agent mobility. One additional advantage of this approach is the fact that, agent migration time can be reduced considerably (due to its limited size).
Constrained mobility is particularly suited to network management tasks that require a relatively long period of time to execute, when dynamic programming of network devices is required or when a large number of data, intended for off-line analysis, is collected by the agent in the remote machine.

2.1.1.1.2 Weak mobility

Weak mobility involves the migration of a mobile agent to a number of machines without preserving execution state information gathered from previous visits. Similarly to constrained mobility, the agent is created and initialised by a client application and is then shipped to an agent host. However, “weak” mobile agents are not confined to this host since they are meant

to perform the same task in more than one network location. Thus, as in weak mobility the mobile agents cannot preserve the information gathered in previous visits, they can only implement tasks in which this information is not required.

A convenient use of weak mobility is situations where management tasks are required to be dynamic decentralised. The advantages of weak mobility approach become obvious, if we consider the case in which a management station has to search for a single value in a table, a data structure typically used to store information inside devices. Particularly, in Simple Network Management Protocol (SNMP) the whole table has to be transferred from the remote element to the management station, where the table rows are searched for this value. Hence, large tables incur heavy unnecessary traffic into the network and result in computational overload on the management station. Alternatively, instead of retrieving this SNMP table remotely by using the typical commands of this protocol (such as get, getnext or getbulk), it is more convenient to send a mobile agent to the node of interest, perform this task locally and report back the results of this tasks to the management station. This prevents the transport of unnecessary data over the network and the purposeless utilization of network resources.

In summary, weak mobility could be considered more suitable for tasks requiring a short duration of time to complete and involving multiple network elements. Furthermore, it is suitable to collect on-line data and perform simple control and configuration tasks from several network elements.

2.1.1.2 Autonomy of Agents

An autonomous agent is an entity situated within its execution environment, senses that environment and acts on it, over time, in pursuit of its own agenda and affecting in this way environment's evolution, future condition and behaviour [6].

The behaviour of an autonomous agent is determined by the analysis of three factors: (a) the environment in which the agent is operating, (b) the task that is designed to accomplish and (c) the design of the agent itself. An autonomous agent is self-determined. It has to monitor the environment and figure out which is the next problem or goal to be addressed. It has to deal with problems in a timely fashion. Typically, it has to deal with many conflicting goals, simultaneously. However, since an autonomous agent is situated in its environment, it is directly connected to its problem domain. The problem domain is typically very dynamic which means that the agent has a limited amount of time to act and that unpredictable events may happen.

2.1.1.3 Mobile agents and Network Management

The mobile agent model provides an alternative to traditionally Client /Server paradigm (among others) because it constitutes a uniform paradigm for distributed systems. In the client/server model execution of code is implemented with various ways such as RPC (Remote Procedure Call) or its similar implementation in java RMI (Remote Method Invocation), or by using CORBA (Common Object Request Broker Architecture) etc. In the client /server paradigm a server advertises a set of services that can be used by potential clients i.e. to access some network resources. The code that implements this functionality is hosted locally by the server, who also provides the execution environment and required resources .The potential clients request then a service or a set of services from this server by using one of the aforementioned methods (RMI, RPC etc).

The mobile agent model is more flexible and robust than the client /server paradigm by having the following characteristics:

- The code or service logic is not bound to any specific location in the network, but is available throughout the network.
- Mobile agents can travel from a network node to another, find the appropriate resources, and execute their task locally (not remotely as in the Client/Server model). This in some cases might improve network performance (as discussed in paragraph 2.1.1.1.2)
- Any host in the network is allowed a high degree of flexibility to possess either the service logic, or resources.

The mobility of agent can be useful in the context of network management, since agents can migrate from node to node and executing their tasks locally. Some other useful properties of agents, that stem from their built-in attributes, are that agents are able to learn and acquire new skills and knowledge and thus becoming capable of performing more tasks than originally have been designed. Network management could be improved by the use of mobile agents since problems encountered in the Client/Server model such as delay, variable delays (jitter), unnecessary network and resource utilization are avoided. Mobile agents are naturally heterogeneous and could be written in any programming language and yet be able to communicate with each other, property extremely helpful in the heterogeneous networks and diverse operating system platforms. Mobile agents, which are naturally robust, can also encapsulate

protocols, adapt dynamically to their execution environment and react autonomously to changes.

As a result of the aforementioned properties, could be concluded that mobile agents offer a model that can be effectively exploited in the context of network performance monitoring system.

2.1.1.4 The Grasshopper Agent Platform

For the implementation of the software mobile agents the Grasshopper agent platform is used and some knowledge about this platform should be given first. Grasshopper is an agent development platform that enables one, to develop and deploy a wealth of distributed, agent-based applications written in the Java programming language. In detail, Grasshopper enables one to:

- Create autonomous acting agents, which are able to migrate.
- Transparently locate agents and send messages to them.
- Manage and control agents during their execution by providing sophisticated tools.

Furthermore, the main elements of the Grasshopper platform are:

- ***Agency***: It provides a runtime environment for agents. Every agency may interact with each other, either through remote communication or agent migration. At least one agency must run on each host in order to support the execution of agents.
- ***Places***: A place provides a logical grouping of functionality inside an agency. In every agency exists by default a place named "InformationDesk". Every agent with no determined place is transported to the "InformationDesk" where it can look for further information.
- ***Region***: It is responsible for the management of agencies and agents. A region consists of one region registry and several agencies.
- ***Region Registry***: It provides information (kind of a database) about agencies, places and agents in one region and is used to locate them.

2.2 Network Performance Monitoring

2.2.1 Active and Passive Measurements

The term passive measurement refers to the process of measuring network parameters, without creating or modifying any traffic on the network [1]. In contrast with passive measure-

ments, active measurements, introduce some specific packets into the network, and these packets are timed as they travel through the network that is being measured.
Passive measurements can provide a detailed set of information about the one point in the network that is being measured. Examples of the information that can be acquired from passive measurements are:

- Traffic / protocol mixes
- Accurate bit or packet rates
- Packet timing / inter-arrival timing
- Average packet loss
- Connection bandwidth

Passive measurement can also offer a means of debugging a network application, by providing a user with the entire packet contents, as seen on the network.
Active systems provide very little information about a single point of a network. They instead give a representation of the characteristics of the entire network path between two hosts. Active systems can provide such indications of a networks performance as:

- Packet round trip time (RTT)
- Delay

Some active systems can also give indications of the following:

- Asymmetric delay times
- Alterations in routing paths between hosts

In the context of this project passive measurements are used through the SNMP protocol, while active measurements are supported via the dispatch and reception of an 'echo' stream. The delay performance parameter is calculated through an active measurement of the round-trip delay involving the execution node and a targeted node. The system at a managed node sends an 'echo' stream to the targeted node and receives it back again as a reply. For this operation the time elapsed is measured in order to obtain the round-trip delay. The value of the round-trip delay is then divided by two in order to obtain a delay result (in msecs). The parameter of IP input rate performance parameter is obtained through passive measurements on the targeted node (packets/sec).

2.2.2 Uniformly Weighted Moving Average (UWME)

A key aspect in network monitoring is to determine utilisation of the managed network. This could be done by reading periodically raw values of some counter objects that count bytes or packets (depending on the network parameter observed) [7]. The difference between subsequent observations is divided by the elapsed time and gives the instantaneous throughput.

In a moving average algorithm, a number of instantaneous observations are used to calculate the throughput in order to smooth-out the value and avoid high fluctuations. Note that the throughput is the derived value of the counter (Dx/Dt). The simplest form is the Uniformly Weighted Moving Average (UWME) in which each instantaneous throughput observation is given the same weight in the calculation of the "throughput mean". The mean value in the UMWE algorithm is calculated as follows:

$$M(t) = \frac{1}{N} \sum_{i=0}^{N-1} V[t - (i * Dt)]$$

Where

- M(t) is the estimate of the mean value at time t
- C(t) is the raw value of the counter at time t
- V(t) is the instantaneous throughput at time t i.e. [C(t)-C(t-Dt)] / Dt
- Dt is the period between successive observations of C(t) and V(t)
- N is the number of V(t) calculations required to estimate the mean

Note that for N calculations of the instantaneous throughput V(t) we need N+1 values of the raw counter attribute. An example mean estimation for N=3 is M(t)=1/3[V(t- 0*t)+ V(t-1*Dt)+V(t-2*Dt)]

Another variation of the aforementioned algorithm is the Exponentially Weighted Moving Average. In this algorithm, more recent observations are more significant (greater exponent assignment) and contribute more to the average.

2.2.3 Severity indicating gauge-threshold model

Another approach to monitor a network resource is achieved with the gauge threshold and severity indicating gauge-threshold model, allowing a managing entity to receive notifications if a threshold value or values have been reached [2]. Severity indicating gauge-threshold model can be regarded as an extension to gauge threshold with which has some commonalities. One of them is the syntax of the severity indicating gauge- threshold, which is similar to the gauge-threshold, but it is enhanced with one optional field, the severity indication parameter. This parameter has two submembers, the notify-high and notify low for each threshold level.

Also a notification switch might be implemented in order to indicate if notifications are wished to be emitted or not. In order for the severity indicating gauge-threshold model to be illustrated, following is provided an example [2] (Figure 1).

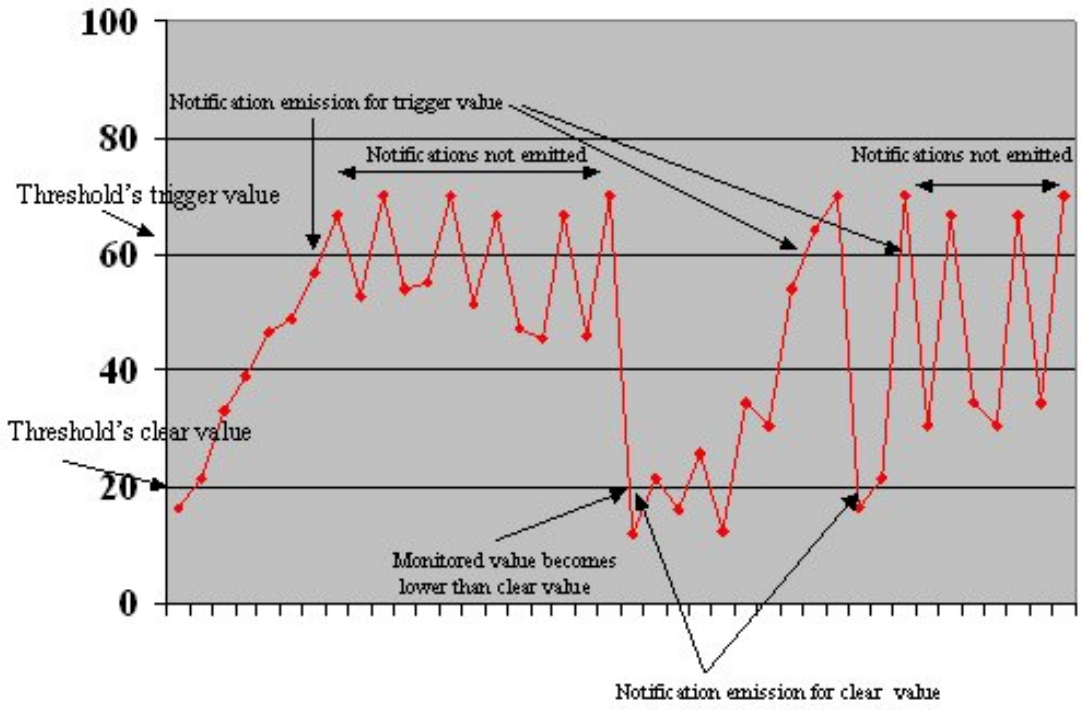


Figure 1 The severity gauge threshold model (with one threshold)

Figure 1 illustrates one threshold level of the severity gauge threshold model, with the arrows showing when severity indications are emitted. The used severity indicating gauge-threshold model starts to emit some notifications when the severity indicating gauge-threshold reaches some certain values (notification switch is assumed that is set to true). The emission of these notifications happen as follows:

- When the gauge value becomes greater than or equal to threshold trigger value, in a *positive going direction*, then the defined event notification is triggered.
- If the gauge subsequently crosses the threshold's trigger value further generation of event notifications shall not occur unless the gauge value becomes less than or equal to threshold clear value.
- When the gauge value becomes less than or equal to threshold clear value, in a *negative going direction* and the gauge value has been greater than or equal to threshold trigger value, then the defined event notification is triggered.

- If the gauge subsequently crosses the threshold's clear value further generation of event notifications shall not occur unless the gauge value becomes greater than or equal to threshold trigger value.

These two threshold values (indicators) of the severity indication parameter are similar and can be regarded as operating reciprocally. Both of them operate exactly the same:

- Initially, emit a notification when the gauge value reaches a predefined value (either upwards or downwards for both cases respectively).
- Each indicator triggers another notification to be sent when the value of the gauge has already reached (less or greater than) the value that triggers the opposite indicator.

The severity indicating gauge-threshold model is implemented in the monitoring processes that the network performance monitoring system performs, in order for the appropriative notifications to be emitted to the management authority and system's user.

3 Work done

3.1.1 Requirements

The main focus of this project is the enhancement of an existing network performance monitoring system with the behaviour needed in order for it to act autonomously to the encountered errors and failures during its operation. The autonomous property of this network performance monitoring system is realised through intelligent software entities that contain the logic not only to behave independently but also to proactively or reactively deal with the errors and faults encountered in the system itself. These software entities sniff the environment changes and execute the necessary procedures or actions that sustain network performance monitoring system operation even when a critical failure occurs. Environment changes may involve a number of issues such as:

- Communication failures amongst software entities.
- Availability of collaborating entities.
- Required updates of management functionality.
- Resource over-utilization and agent migration issues.
    - Processing load balancing in order to avoid resource over-utilization.

- Agent migration in conjunction with data volume.
- Agent migration in conjunction with the available network resources.

An Intelligent Adaptation to the Environment (IAE) Agent-Based Network Management System (ABNMS) contains software entities that monitor and provide the following functions:

1. Network Management tasks such as network resource utilization, delays, and network work balancing etc.
2. Collaboration of software entities in order to achieve the management tasks in the most efficient manner.
3. Communication of mobile software entities, taking advantage of their mobility to execute complex and resource demanding management tasks in any network node.
4. Migration of software entities to network nodes so as providing them with the required logic to perform management tasks on demand.
5. Reactiveness to network changes, in order for network failures to be dealt gracefully.
6. Communication failures amongst local or remote communicating software entities are detected and handled.
7. Notification of the management entities, about current or imminent network failures, network node (un)-availability etc.
8. Update of management logic to the software entities with new management functionality.
9. Any software entity involved in a network management task should exhibit some important environment-related properties and be able to handle and react to a number of specific system failures such as:
    a. Proximity:
        i. Management logic should execute, as close to the required resources as possible, ideally logic and resources should share the same host.
        ii. When a migration is not feasible, a network node that is ‘closest’ to the desired node may be used to accommodate this task.
    b. Availability of
        i. Collaborating network entities. The system provides the solutions for problems that may occur in the collaborations of these entities.
        ii. The required management logic. The system should be able to lookup, decide and send to the desired node the required management logic.

iii. A required management resource. The system should be capable of identifying resource availability, undertaking necessary actions in order to inform the management authority and finally try to resolve this fault.
iv. Operational Resources. This includes any resources at a network node required for the execution of a software entity task.

10. The management system has an internal mechanism so that software entities report their status, failures, problems and malfunctions to the appropriate recipient.
11. An Intelligent Adaptation to the Environment (IAE) Agent-Based Network Management System (ABNMS) should attempt to resolve any problem by its own, find an appropriate fall back solution and give up only when the task is impossible within reason (but taking care to report of the situation first).

3.1.2 Specifications

Before presenting the specifications of this project, it is considered as appropriate firstly to be mentioned the aspects of the performance monitoring system, which the implementation of this project has been based upon (implementing, reusing and extending its capabilities and features as described in the previous paragraph).

3.1.2.1 The previous network performance monitoring system

The former network performance monitoring system provided some kind of "autonomous behaviour" of "intelligent" software agents in the context of performance monitoring of QoS-enabled IP networks. The system performed tasks such as network resource utilization, delays, and network work balancing. The system took actions autonomously on behalf of the user and performed the management tasks, based on the analysis of performance information gathered. Through this analysis it fine-tuned its task by having adjusted parameters of its operation and provided sophisticated reports of performance information to the user. The following section aims to provide a brief description of the previous management system.

3.1.2.1.1 Previous System Capabilities

The previous system performed the following functionalities:

- *Autonomous selection of migration node*. Mobile agents would be able to choose autonomously the most efficient destination node (and the least overloaded one) to migrate in order to perform a task.
- *Autonomous pro-active behaviour based on current results*. The system would check on current performance behaviour in comparison to applied thresholds and when a given rule was met, would send notifications (i.e. the threshold values at which to report alarms). The system would be able to take actions in order to detect errors or lapses in the proper performance of the system and finally would have isolated and solved them before they adversely impacted network performance.
- *Initial configuration of the monitoring system parameters*. Some parameters would be initialised and based on available network resources and user's Quality of Service (QoS) request based on service requirements.

3.1.2.1.2 Existing Performance Monitoring Agents

The previous network management System was composed of the following agents:

- Master Agent (MA): A stationary agent that initiated a management task and handled interactions with the user or client application (i.e. was responsible for the coordination of the performance monitoring system).
- Target Agent (TA)**:** A stationary agent that pre-existed at the targeted network element, allowing Worker agents to access a number of required network element resources.
- Worker Agent (WA**)**: A mobile agent equipped with the required management logic, migrating and executing its task in the appropriate remote node. Any reports or notifications generated were sent from the Worker to the Master agent. This agent was composed of some parts and each time–depending on the task, Worker loaded any of these components. Each part had also some sub-modules each of one responsible to accomplish a specific task. The existing modules of the Worker agent were:
    - Network Management Module (NMM). Involved the conventional and autonomous aspects of a specific management task.
    - Carrier Module (CM) Responsible for the initial migration or any subsequent re-location of the agent.

It should be mentioned that these sub-modules of the Worker agent were not discrete, which means that the functionality of the Network management module and carrier module existed as entities but they were embedded into Worker's code.

In the implementation of this project the functionalities of the Worker agent are separate, meaning that the sub modules of the Worker that have been developed are separate java classes. With this approach, the system is easier to be illustrated, maintained, extended and understood.

3.1.2.2 Intelligent Adaptation to the Environment Module (IAEM)

In this project the network management functionality is extended and the required intelligence (system's autonomous capabilities) is added in order for failures and problems to be dealt and solved autonomously by agents. The system detects these failures and problems, analyses them, finds the appropriate solution (or take fallback measures), and finally takes suitable actions to solve them.

The network management system is reactive to network changes, so that when network failures are detected, are dealt quickly and addressed effectively. These failures may include communication failures amongst local or remote communicating software entities.

The system considers and deals with software entities proximity related problems regarding migration denials, unavailable execution environment, and network node failures.

The network management agents are able to send notifications to the appropriate management entities, about network failures, network node unavailability etc so that the necessary autonomous actions to be taken.

Finally, the system handles situations where one or more resources, is/are unavailable (for instance, due to a router failure).

3.1.2.2.1 IAEM Sub-Modules

The IAEM is composed of three separate sub-modules, each one responsible for performing different tasks. These sub-modules are:

- **Controller.** This sub-module receives the task requests from the Network Management Module (which resides within the Worker agent), and performs the appropriate actions in order to carry out these tasks.
- **View.** Handles the outgoing communications from the Intelligent Adaptation logic (IAL) sub-module, to the other modules of the system (such as the Network Management Module, the Diagnostics Agent, the Repository agent etc). It communicates with

the network management entities (Master or User Agent) in order to report a problem. Finally it communicates with the Carrier Module so as ordering the latter to start the process of Worker's agent migration.

- **Intelligent Adaptation Logic (IAL) (Model).** This sub-module provides the required logic to handle a failure. It decides which solution would be given, calls and loads the appropriate logic, and provides the required results. It keeps track of task progress and notifies if needed (through View sub-module) the appropriate network entities (e.g. Master Agent, or the system Administrator). It constitutes the heart of the IAEM, since manipulates, coordinates and tasks all the other sub-modules of the system.

The IAEM module is created from scratch. The other existing agents (Master –Target- and Worker agent) have been integrated accordingly in order to accommodate the extra functionalities, interfaces and responsibilities. Further more to the sub modules of the Worker, in this project three additional entities have been constructed from scratch. These entities are three agents, named Repository Diagnostics and Creation agent with the following responsibilities:

- **Diagnostics agent.** The task of this agent is to perform active and passive measurements (as described in paragraph 2.2.1) in collaboration with the Target agent(s). This task is initiated from the IAL module, through the View module, in order for a failure to be identified, and, as the name of the agent implies, diagnosed. The diagnostics agent returns as a response to the request of the IAL, the diagnosis of the failure (i.e. if it is a true failure or not). Moreover, the capabilities of diagnostics agent could be also used by the IAL module in order for the latter to periodically check if an error /failure is still observed on the network.

- **Repository agent.** The task of this agent is the provision of a software entity that has failed. This task is initiated from IAL module with the dispatching of a creation request to the Repository agent. Thus, the functionality of this agent is to provide either a correspondent /collaborator (i.e. the code of the Master agent or Target agent respectively) or a logic update/creation in case that some of these failures are encountered.

- **Creation agent**. This agent is created by the Repository agent and then is dispatched to the desired network node in order to create there locally the requested software en-

tity. After the successful entity creation, removes itself from the execution environment.

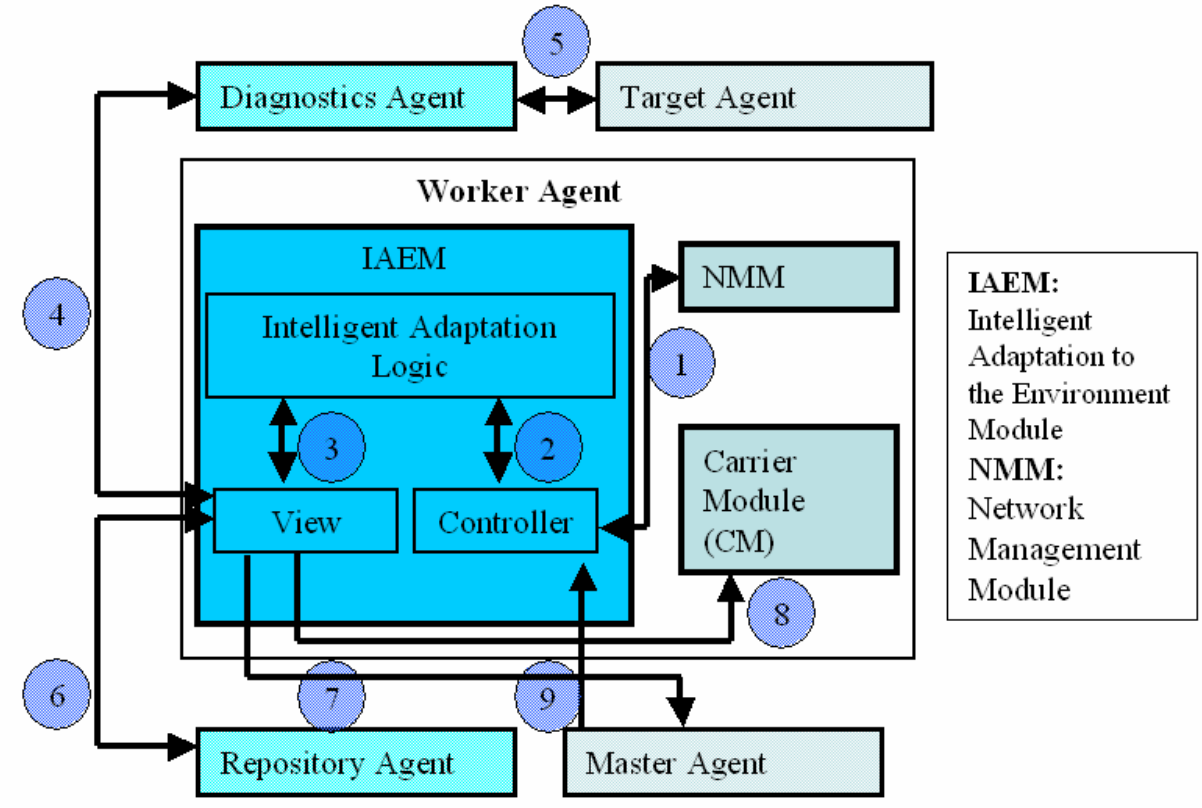


Figure 2 Worker's sub modules and their interfaces

Figure 2 shows Worker's agent sub-modules and the interfaces among them. Each number in figure 2 represents the relevant interface between the two entities that is connecting (and correspondents also to the numbering of Table 1). These interfaces have the following names and functionalities.

	Interface name	**Functionality**
1	IExControllerNetMang	Maintained by the Controller sub -module in order to receive requests from WorkerExtentionAgent (status changes and fault notifications).
2	IInControllerAdaptLogic	Maintained by the Adaptation Logic sub –module in order to receive requests from the Controller sub-module
3	IInAdaptLogicView	Implemented by the View sub –module, so that Intelligent Adaptation Logic sub –module to send requests to that.
4	IExViewDiagnos	Implemented by Diagnostics agent and used for the required communications between Diagnostics agent and

		View sub–module.
5	IExDiagnosTarget	Implemented by Target agent and used for the communications between Diagnostics agent and Target agent.
6	IExViewRepos	Implemented by Repository agent and used for communications between View sub–module and Repository agent.
7	IExStatusReportView	Implemented by any entity interested in receiving such messages.
8	IExViewCarrier	Implemented by Carrier sub-module in order for it to receive migration commands from Intelligent Adaptation Logic through View sub–module.
9	IExMasterController	Implemented by the Controller sub –module and used by the Master agent in order to send new tasks to Worker.

Table 1 Description of interfaces used in the adaptation system

3.1.2.2.2 Commonalities of system failures handled.

The following table contains the failures and errors that the IAEM is capable of handling

	Type	Case	Solution	Fall back	Side effect
1	Proximity	Migration denied	1.Identify failure from Diagnostics agent 2. Migrate to a nearby alternative node.	Operate from node of creation.	Inform managing entity, periodically reattempt.
2		Runtime Environment Unavailable	1.Identify failure from Diagnostics agent 2.Migrate to a nearby alternative node.	Operate from node of origin.	Inform managing entity, periodically reattempt.
3		Network Node Inaccessible:	None	None	Inform managing entity, periodically reattempt

4	Unavailability of	Collaborator	Request for a collaborator from repository agent	None.	Inform managing entity.
5		Correspondent	1.Identify failure from Diagnostics agent 2. Request for a correspondent from repository agent	None	Inform managing entity and user.
6	Remote Communication Failure	Remote Communication Failure	Identify failure from Diagnostics agent	Continue operation, retain information intended for communication.	Inform managing entity, periodically reattempt.
7	Resource Availability	Resource Availability	Identify failure from Diagnostics agent	None.	Inform managing entity, periodically reattempt.
8	Availability of Logic	Logic Unavailable:	Request for management logic modules from repository agent	None.	Inform managing entity.
9		Logic Update Required:	Suspend and request for management logic modules from repository agent	Continue operation as is.	Inform managing entity.
10	Availability of Operational Resources	Critical Resource Utilization	1.Identify failure from Diagnostics agent 2.Release some resources.	Migrate to a nearby alternative node.	Inform managing entity, if fall back is followed periodically check resources for a migration back

Table 2 System failures handled by IAEM

All these failures and errors have some common characteristics as far as the actions that IAEM has to perform, are concerned. These are:

All failures and errors are sensed and identified by the Worker agent. In case of an error or failure, the same agent, immediately changes its state, in order to reflect this error to the system. The IAEM module periodically checks the Network Management module (located within the Worker agent) status so as realising if an error has occurred. Then the Controller of IAEM gets from NMM the error/failure type and starts the adaptation procedure.

Most of the solutions have as a result one of the following actions to be taken:

- A migration of Worker agent to another host
- A request from IAEM for an entity to be created
- An update /creation of a management logic entity.

The cases 1 and 2 of Table 2 are similar as far as the actions to be performed by IAEM are concerned. The same applies to the cases 4 and 5, with the divergence that a different entity is requested to be created. These two cases are slightly diverse, since in case 5 a request to Diagnostics agent is made in order for the failure to be verified whereas in case 4 this is not required.

Most of IAEM procedures end with a notification been sent either to Master agent or to the user.

3.1.3 Design

3.1.3.1 Class diagrams

The class diagrams of the entities used in this project are presented in this section. In the following two figures the interfaces and the objects are depicted. The functionality and description of the interfaces have been discussed in section 3.1.2.2.1, whereas IAEM's sub modules, Diagnostics, Creation and Repository agents' functionality have been also introduced in section 3.1.2.2.1.

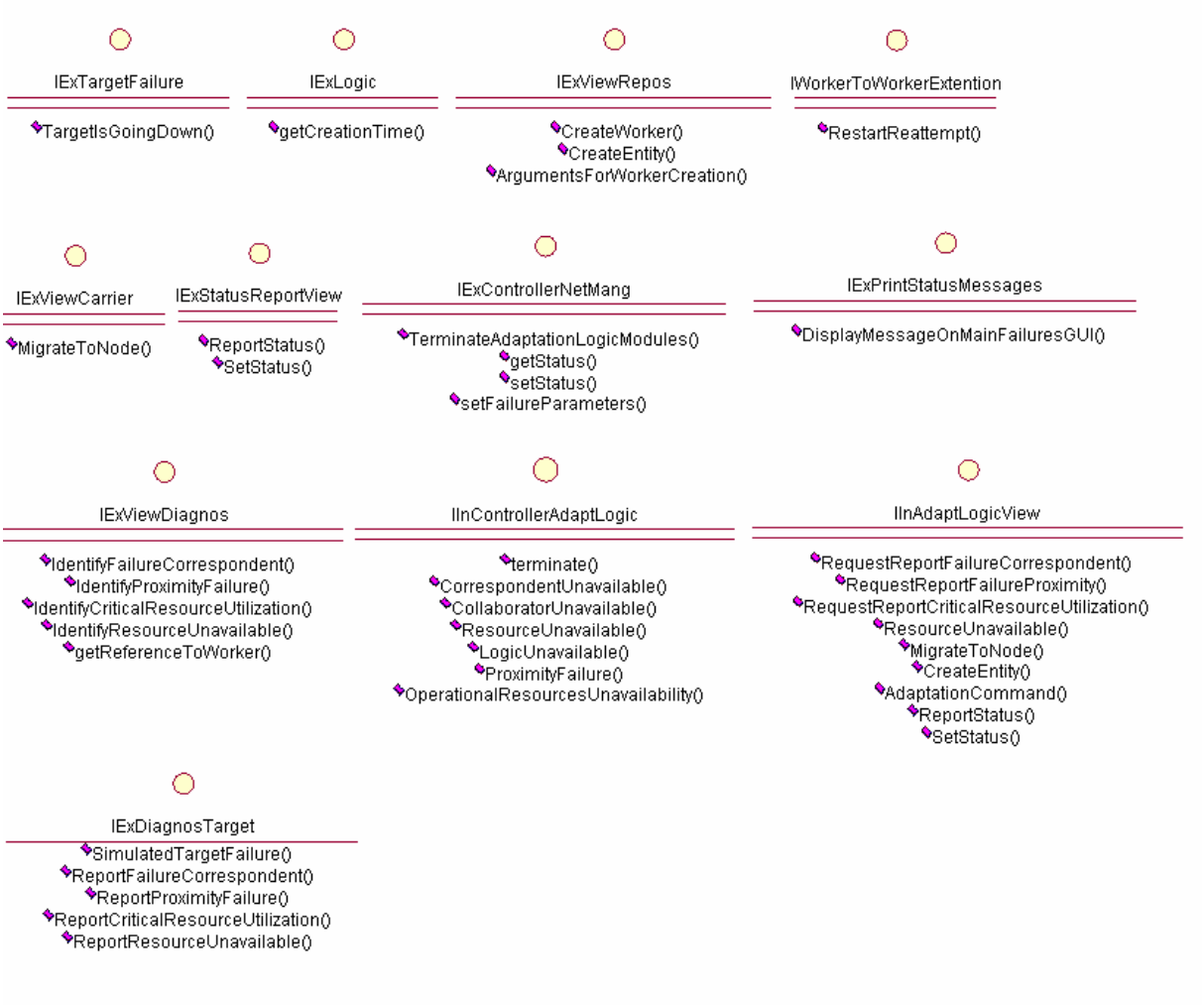
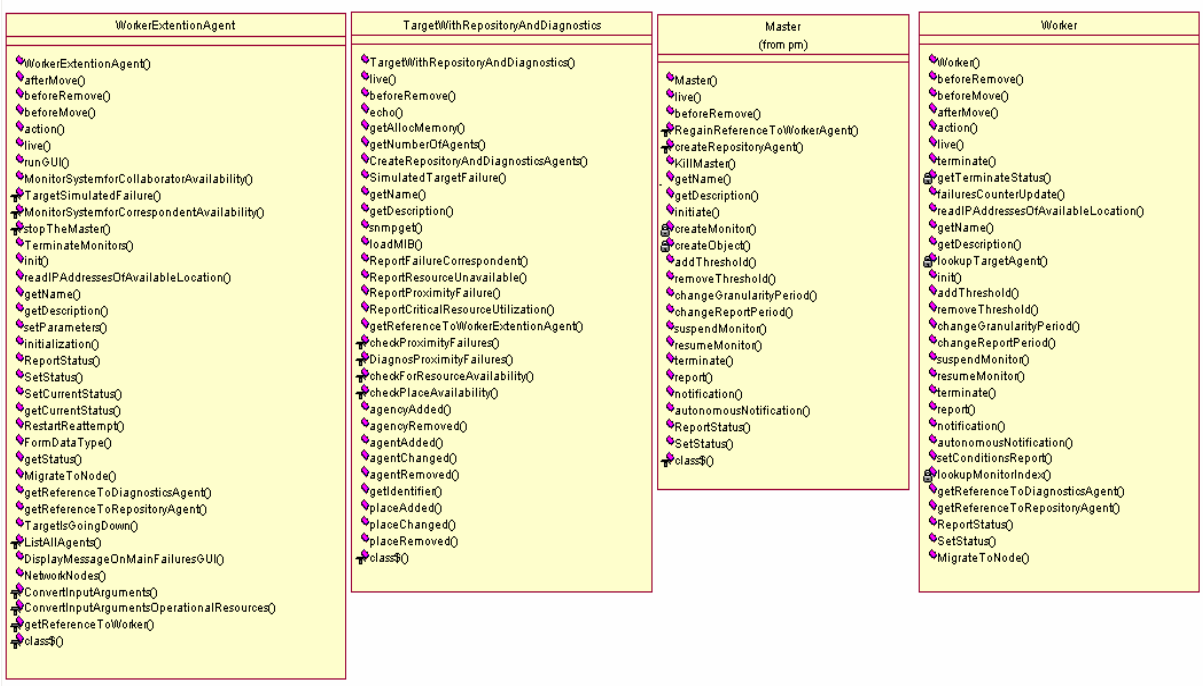


Figure 3 Interfaces class diagrams



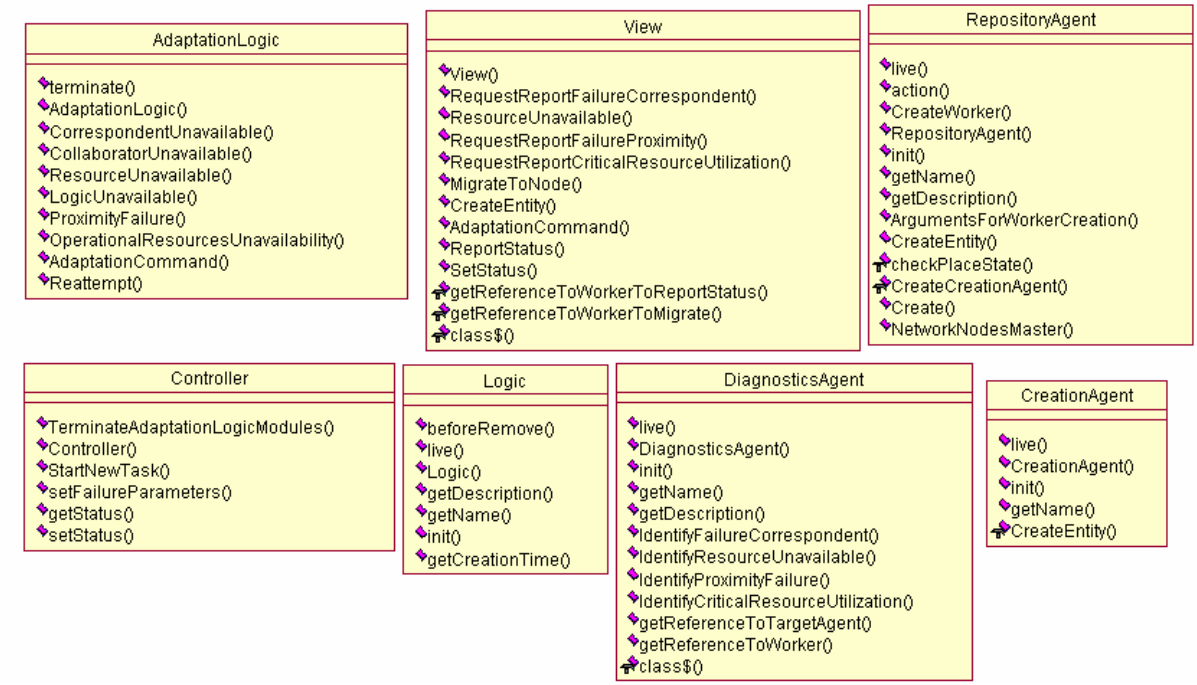


Figure 4 Project's objects

The data types used in the implementation of this project are presented in the following in the following figure.

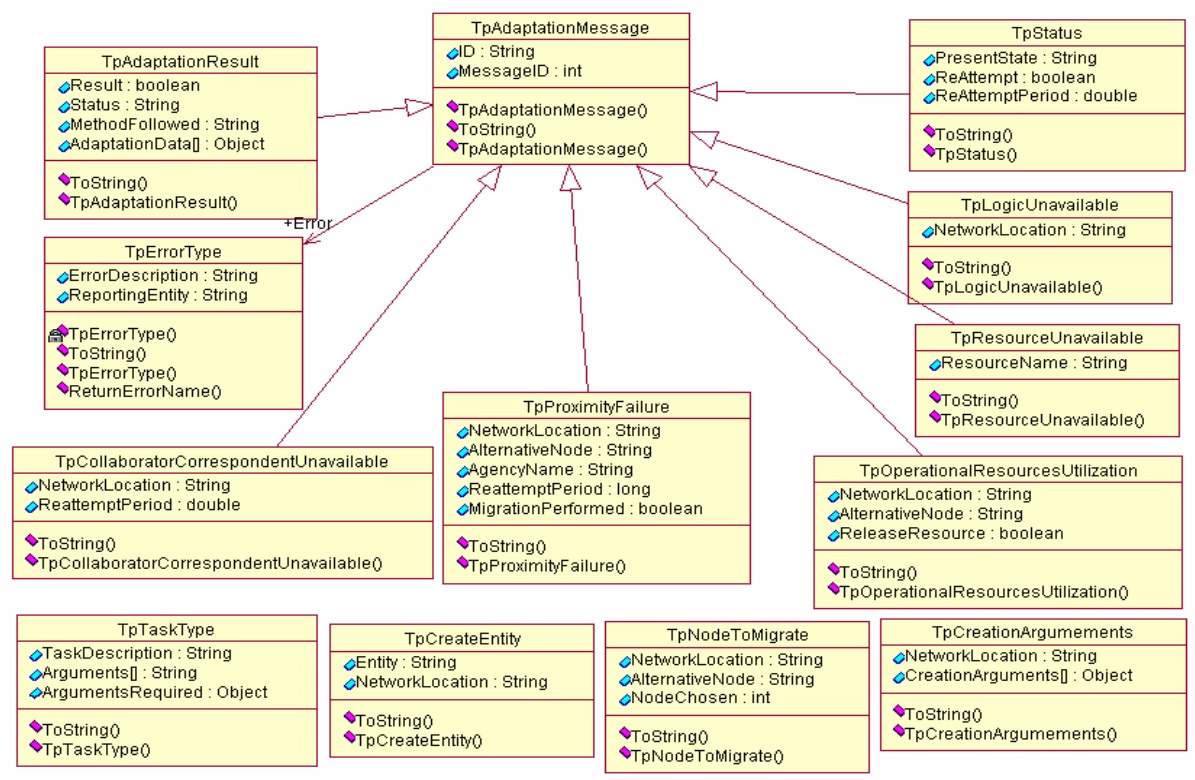


Figure 5 Project's data types

The interfaces and their available methods are presented in the following table.

Interface name	Methods	**Input**	**Return type**
IExDiagnosTarget	**ReportFailureCorre-spondent**	(int RequestID, TpCollaboratorCorresponden-tUnavailable Message)	TpAdaptationResult
	ReportProximityFailure	(int RequestID, TpProximityFailure Message)	TpAdaptationResult
	ReportCriticalResour-ceUtilization	(int RequestID, TpOperationalResoursesAnavail-able Message)	TpAdaptationResult
	SimulatedTargetFailure	null	void
IExControllerNetMang	**TerminateAdapta-tionLogicModules**	null	void
	getStatus	null	Object
	setStatus	(String status)	void
	setFailureParameters	(Object Data, String Datatype, IExViewDiagnos ReferenceTo Di-agnostics, IExViewRepos Refer-enceToRepository)	void
IExViewCarrier	**MigrateToNode**	(int RequestID, Object Data, String Datatype, TpNodeToMigrate Mes-sage)	void
IExViewDiagnos	**IdentifyFailureCorre-spondent**	(int RequestID, TpCollaboratorCorresponden-tUnavailable Message)	TpAdaptationResult
	IdentifyProximityFailure	(int RequestID, TpProximityFailure Message)	TpAdaptationResult
	IdentifyCriticalResour-ceUtilization	(int RequestID, TpOperationResourcesUtilization Message)	TpAdaptationResult
	getReferenceToWorker	null	AgentInfo[]
	IdentifyResourceUn-available	(int RequestID, TpResourceUnavailable Message)	TpAdaptationResult
IExStatusReportView	**ReportStatus**	(TpStatus Message)	void
	UpdateStatus	(TpStatus Message)	void
IExViewRepos	**CreateEntity**	(int RequestID, TpCreateEntity Message)	boolean

IInAdaptLogicView	**RequestReportFailure-Correpondent**	(int RequestID, TpCollaboratorCorrespondent Unavailable Message, IExViewDiagnos ReferenceTo Diagnostics, IExViewRepos ReferenceToRepository)	TpAdaptationResult
	RequestReportFailure-Proximity	(int RequestID, TpProximityFailure Message, IExViewDiagnos ReferenceTo Diagnostics, IExViewRepos ReferenceToRepository)	TpAdaptationResult
	RequestReportCritical-ResourseUtilization	(int RequestID, TpOperationResourcesUtilization Message, IExViewDiagnos ReferenceTo Diagnostics, IExViewRepos ReferenceToRepository)	TpAdaptationResult
	MigrateToNode	(int RequestID, Object Data, String DataType, TpNodeToMigrate Message, IExViewDiagnos ReferenceTo Diagnostics, IExViewRepos ReferenceToRepository)	void
	CreateEntity	(int RequestID, TpCreateEntity Message, IExViewRepos ReferenceToRepository)	TpAdaptationResult
	ResourceUnavailable	(int RequestID, TpResourceUnavailable Message, IExViewDiagnos ReferenceTo Diagnostics, IExViewRepos ReferenceToRepository)	TpAdaptationResult
	AdaptationCommand	(int RequestID, Object Message)	TpAdaptationResult
	ReportStatus	(TpStatus Message, IExViewDiagnos ReferenceTo Diagnostics)	void
	SetStatus	(TpStatus Message, IEexView Diagnos ReferenceTo Diagnostics)	void

IInControllerAdapt-Logic	**Terminate**	null	void
	CorrespondentUnavail-able	(int RequestID, TpCollaboratorCorresponden-tUnavailable Message, IExViewDi-agnos ReferenceTo Diagnostics, IexView Repos ReferenceToRe-pository)	void
	ResourceUnavailable	(int RequestID, TpOperation Re-sources Utilization Message IEx-ViewDiagnos ReferenceTo Diag-nostics, IExViewRepos Reference-ToRepository)	void
	CollaboratorUnavailable	(int RequestID, TpCollaborator CorrespondentUnavailable Message IExViewDiagnos ReferenceTo Di-agnostics, IExViewRepos Refer-enceToRepository)	void
	LogicUnavailable	(int RequestID, TpLogicUnavailab le Message, IExViewDiagnos Ref-erenceTo Diagnostics, IExViewRe-pos ReferenceToRepository)	Void
	ProximityFailure	(int RequestID, TpProximityFailure Message, IEx-ViewDiagnos ReferenceTo Diag-nostics, IExViewRepos Reference-ToRepository)	void
	OperationalResource-sUnavailability	(int RequestID, TpAdaptationResult Message, IExViewDiagnos ReferenceTo Di-agnostics, IExViewRepos Refer-enceToRepository)	void
IExLogic	**getCreationTime:**	null	Date
IWorkerToWorkerEx-tention	**RestartReattempt**	(int RequestID, Object Data, String Datatype)	void
IExTargetFailure	**TargetIsGoingDown**	(AgentInfo TargetAgentInfo)	void
IExPrintStatusMes-sages	**DisplayMessageOn-MainFailuresGUI**	String MessageToDisplay	void

3.1.3.2 Sequence diagrams

In this section the sequence diagrams of the failures that the IAEM is able to handle together with a description of the actions involved in each failure, are introduced.

3.1.3.2.1 Proximity errors

3.1.3.2.1.1 Migration Denied and Runtime Environment Unavailable failures.

The first action that has to be taken, in all failures described bellow, is the Worker to identify a system failure and change its status in order to reflect that to the system. It also gathers the required information that describes this failure (e.g. network node, type of failure, entity that first identified this failure etc), which is useful to IAEM in resolving this failure. The Network Management sub-module of Worker agent performs this operation. The Controller sub-module of IAEM, which periodically questions the status of the Worker (with the interface *IExControllerNetMang*), realises Worker's status change and starts the procedure to remedy this failure.

The Controller with the interface *IInControllerAdaptLogic* transfers the information gathered from Network Management module, to the Intelligent Adaptation Logic sub-module of IAEM (Figure 6). The latter then after having processed this information, asks from the Diagnostics Agent to verify this failure. In order to do that, firstly it uses the interface *IInAdaptLogicView* of View sub -module invoking, in this particular case, the method *RequestReportFailure Proximity ()*. Then the View sub –module transfers this request to the Diagnostics agent with the interface *IExViewDiagnos* and its method *IdentifyProximityFailure ()*. Following the Diagnostics agent uses the interface *IExDiagnosTarget* and its method *ReportProximityFailure()* in order to communicate with the Target agent. The latter subsequently verifies or not the failure. Then Diagnostics decides that indeed the failure exists and sends back to the Intelligent Adaptation Logic sub-module a message that states if the failure still continues or not. The messages exchanged in this procedure are all synchronous. This means that the entity, which invokes a method to another entity, suspends its operation and waits until getting the return value of that method invocation.

Finally, the Intelligent Adaptation Logic module receives a response from the Diagnostics agent and if the failure continues, then orders the Carrier Module to migrate the Worker agent to a nearby alternative node, or if a fall back is followed, the Worker agent operates from the node of creation. The Adaptation logic module starts this procedure with the interface *IInAdaptLogicView* of View module and its method *MigrateToNode()*. The View module then with the interface *IExViewCarrier* of the Carrier sub module and its method *MigrateToNode()* instructs the latter to migrate the Worker agent to a nearby alternative node, or if a fall back is

followed then the Worker agent operates from the node of creation. Finally the View module with the interface *IExStatusReportView* that is maintained from Worker agent, informs the latter about the status change. This is done with one of the two methods of *IExStatusReport-View* interface, *ReportStatus(), SetStatus(),* depending on the type of failure and the strategy followed to remedy this failure.

Finally, the Intelligent Adaptation logic module periodically re-attempts to migrate the worker agent to the desired node by using its Reattempt*()* method. This method includes the periodically request of a report from the Diagnostics agent, in order to be informed if the failure persists, and if not to proceed to further actions.

3.1.3.2.1.2 Network Node Inaccessible.

In this case the only action that IAEM performs is to inform the managing entity. In order to do that, Intelligent Adaptation Logic module transfers the result of the failure processing to View mode with IInAdaptLogicView interface. Then the View with the interface *IexStatus-ReportView,* informs Master and Worker agent. The Intelligent Adaptation Logic module periodically invokes its Reattempt*()* method as discussed earlier in order to inspect if the network node is still inaccessible.

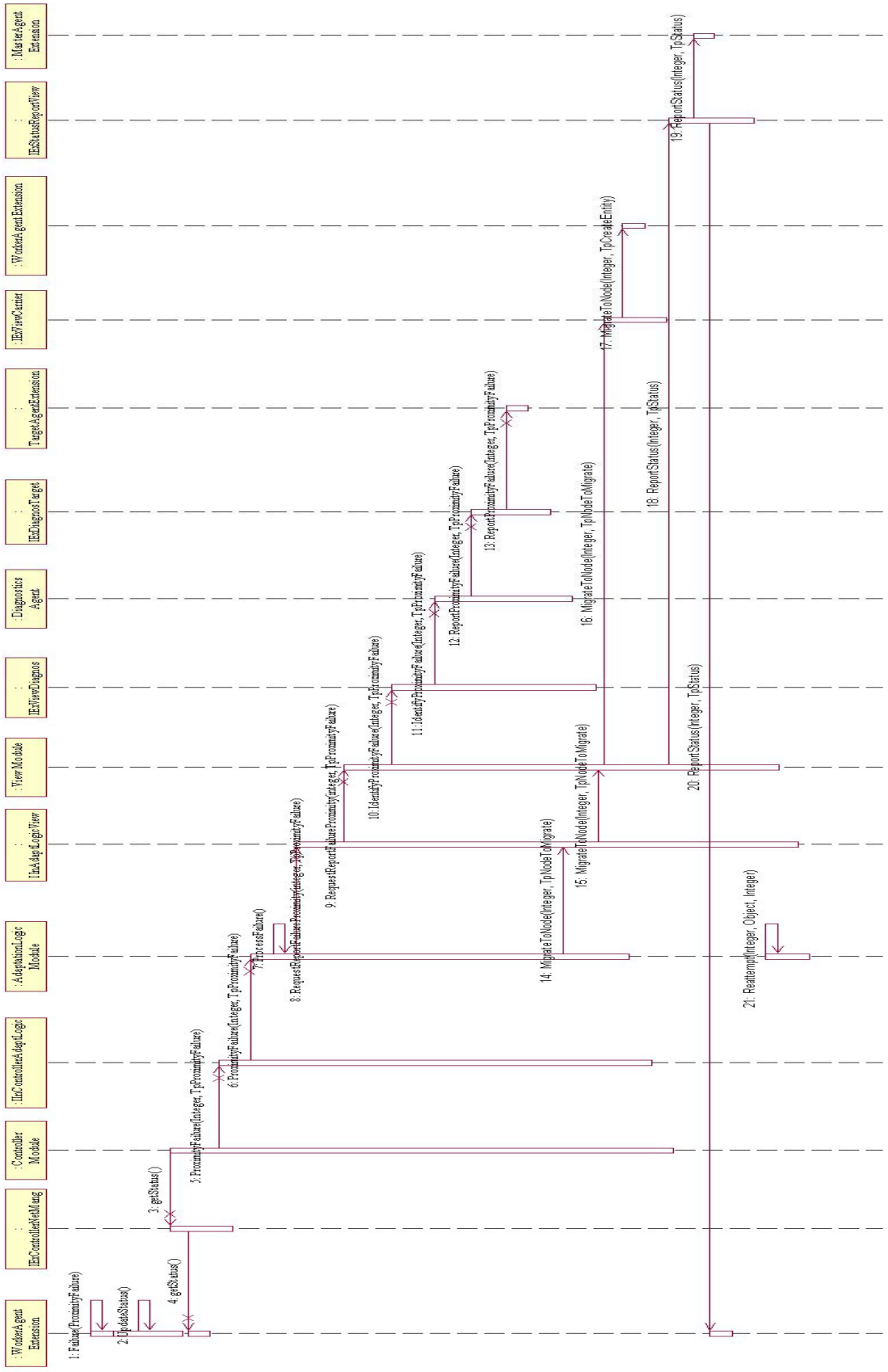


Figure 6 Proximity failure sequence diagram

3.1.3.2.2 Unavailability of Correspondent /Collaborator, Resource Unavailability, Remote Communication Failures.

3.1.3.2.2.1 Correspondent / Collaborator failures.

The failure handling of this case starts after a notification of these failures has reached the Adaptation logic module through its interface *IInControllerAdaptLogic* and its methods *CorrespondentUnavailable() or CollaboratorUnavailable()* respectively(Figure 7). The Adaptation logic module having processed the data describing the failure, in the first case, (Correspondent unavailable) invokes a verification procedure from the Diagnostics agent (as described in the section 3.1.3.2.1.1) with the difference that the methods of interfaces *IInAdaptLogicView, IExViewDiagnos, IExDiagnosTarget* are respectively the followings:

IInAdaptLogicView. RequestReportFailureCorrepondent()
IExViewDiagnos. IdentifyFailureCorrespondent()
IExDiagnosTarget. ReportFailureCorrespondent()

The Intelligent Adaptation Logic module receives failure verification from Diagnostics agent. It should be noted that the failure verification procedure is not needed in case of Collaborator Unavailable failure (Figure 8).
The following actions are common to the two types of failures. The Adaptation Logic module with *IInAdaptLogicView* interface and its method *CreateEntity()* transfers its decision to create a correspondent or collaborator accordingly to the View module. Then, the latter with the interface *IExViewRepos* and its method *CreateEntity()* instructs the Repository agent to create the appropriate entity. Subsequently, creation and migration to the desired network node of a suitable collaborator/correspondent follow. This is performed with the aid of Creation agent that is created for this purpose from Repository agent. Finally the IAEM informs the Master and Worker agent with the *IExStatusReportView* interface and in case of correspondent unavailability, the IAEM also informs the user.

3.1.3.2.2.2 Resource Unavailability

In this case the Intelligent Adaptation Logic module receives a notification, that a network resource is not available, from the interface IInControllerAdaptLogic and its method ResourceUnavailable() (Figure 9). The IAL module then invokes the failure verification procedure from Diagnostics agent by using, in sequence, the following interfaces and methods.

IInAdaptLogicView. RequestReportCriticalResourseUtilization()
IExViewDiagnos. IdentifyCriticalResourceUtilization()
IExDiagnosTarget. ReportCriticalResourceUtilization()

As it can be identified from the above, Resource unavailability and Critical Resource utilization failures are treated with the same method invocations between Intelligent Adaptation Logic –View, Diagnostics and Target entities (failure verification procedure). In both failures the same remedial strategy is followed, because both failures are somewhat similar. The only difference is that in resource unavailable failure, after failure verification procedure, the Intelligent Adaptation Logic sub-module re-invokes the same process after some time has elapsed. On the other hand, in critical resource utilization failure, after failure verification procedure, the Adaptation Logic sub-module either decides the migration of Worker agent to another network node, or the realise of some resources (Figure 11).
In resource unavailable failure, finally the Adaptation Logic informs the Master agent and Worker agent with the *IExStatusReportView* interface and periodically invokes its Reattempt() method.

3.1.3.2.2.3 Remote Communication Failure

This case is similar to the first part of Proximity failures since the Intelligent Adaptation Logic module invokes failure verification from Diagnostics agent. If the failure is verified then no further action is taken and the IAL module tries to continue Worker's operation, and retain information intended for communication. Finally, the IAL module periodically reattempts to establish the remote communication by checking the remote entity. This is achieved by periodically invoking its *Reattempt()*, which in sequence requests failure verification from Diagnostics agent.

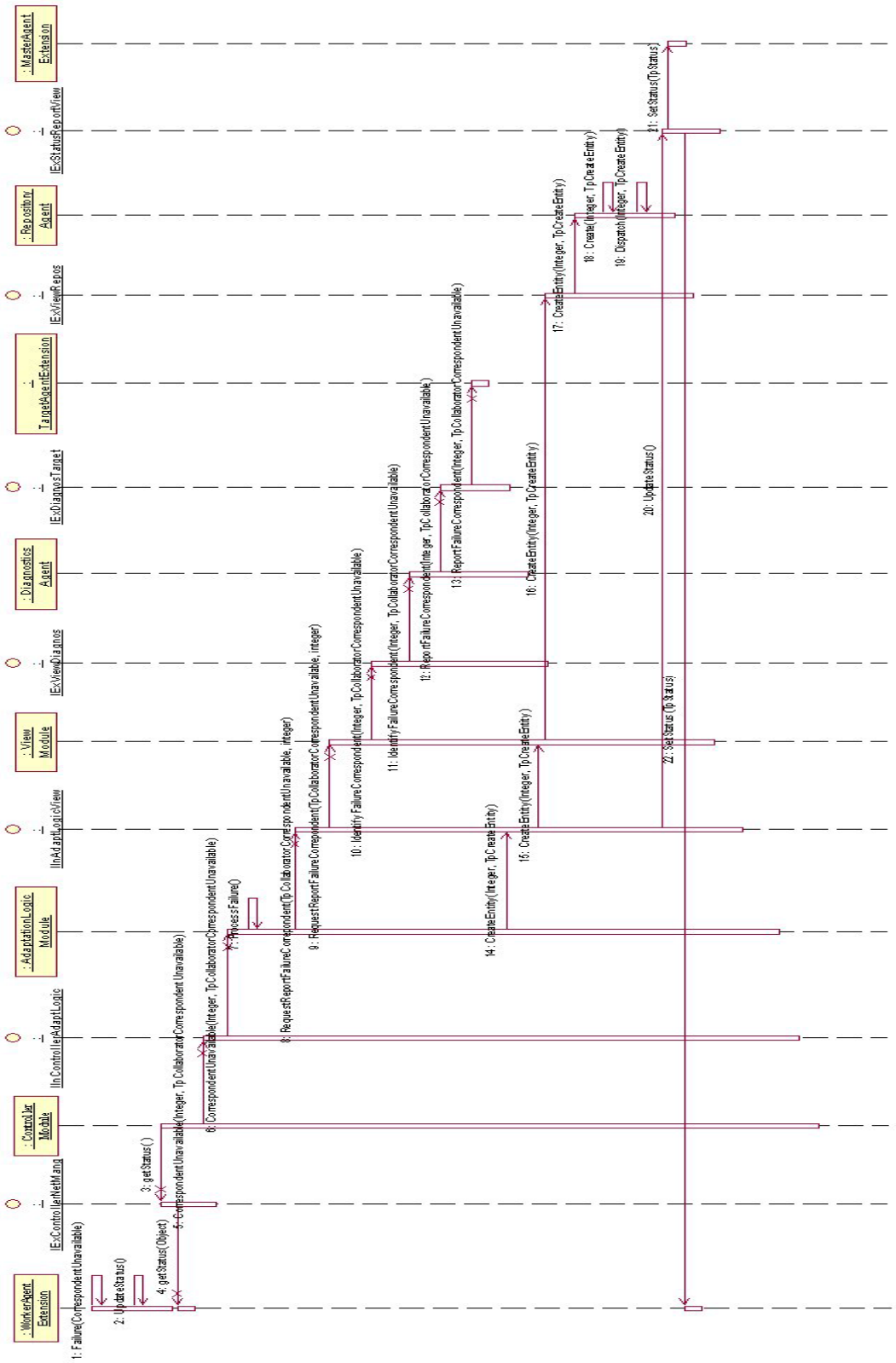


Figure 7 Correspondent unavailable sequence diagram

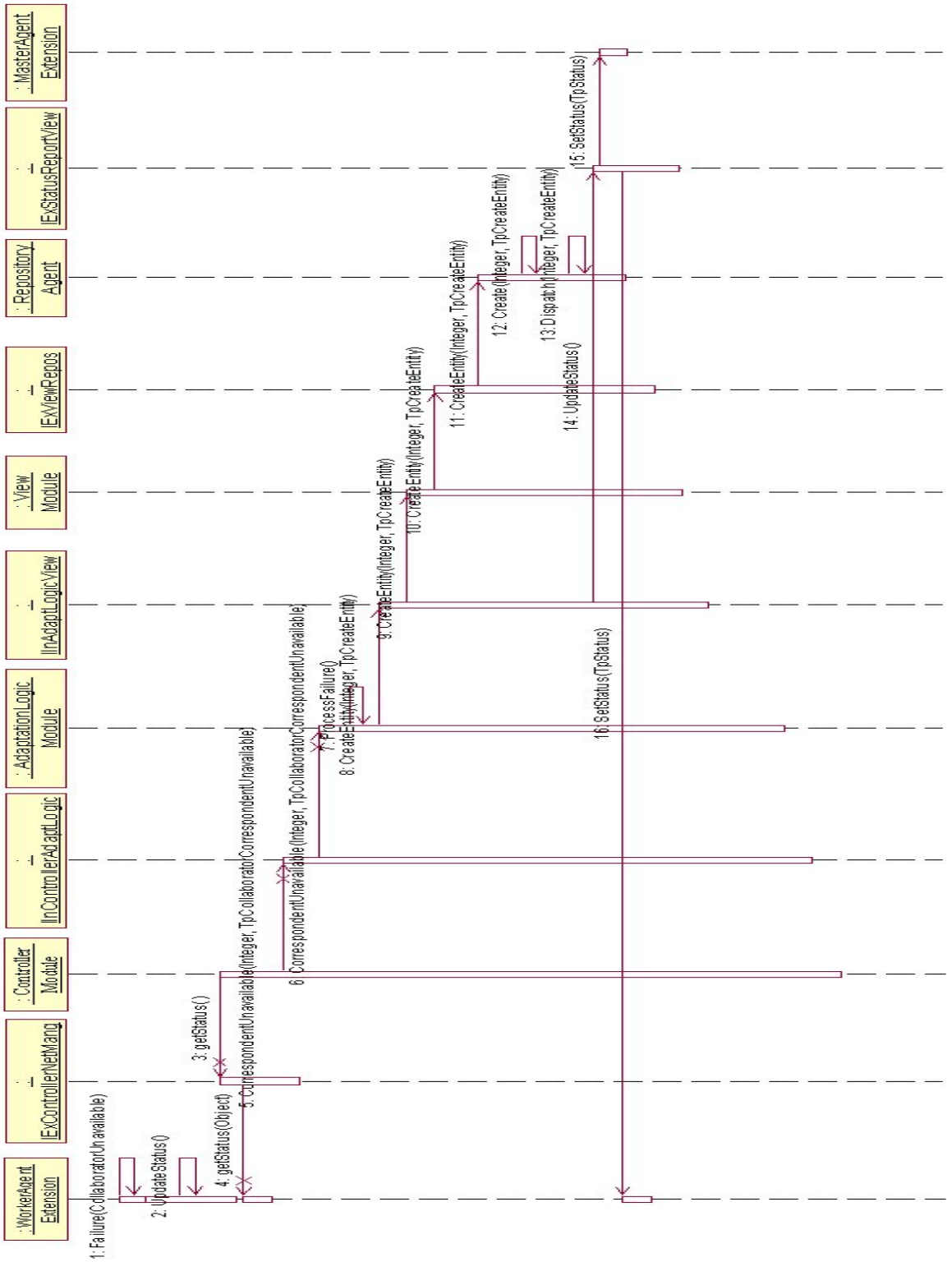


Figure 8 Collaborator unavailable sequence diagram

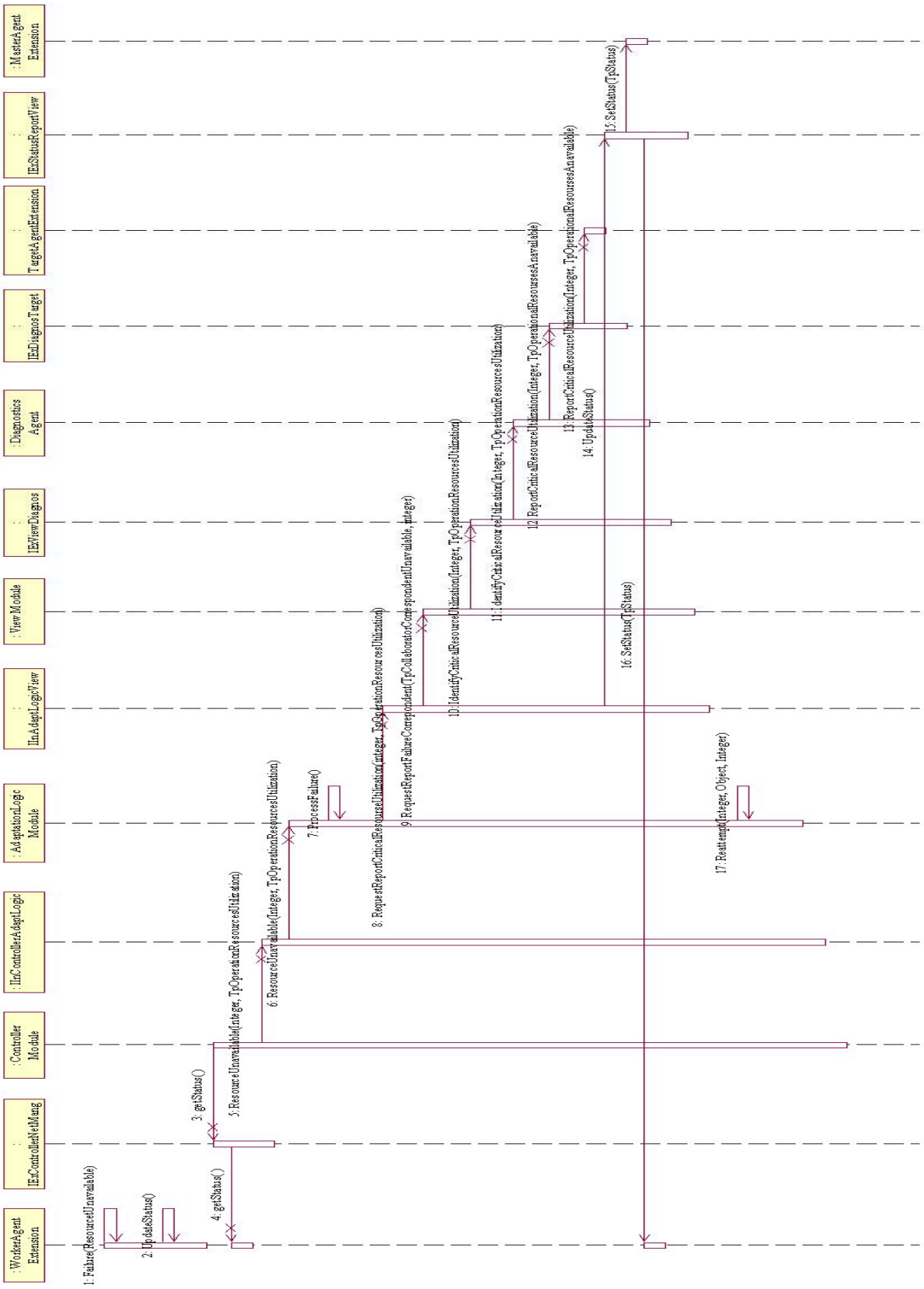


Figure 9 Resource unavailable sequence diagram

3.1.3.2.3 Availability of Logic

3.1.3.2.3.1 Logic Unavailable.

In this case the Intelligent Adaptation Logic module receives the failure notification from the Controller module with the interface *IInControllerAdaptLogic* and its method *LogicUnavailable ().* The IAL module after processing the available data, describing the failure, requests for management logic module creation from the Repository agent (Figure 10). This is performed with the invocation of the following interface/methods of View module and Repository agent:

IInAdaptLogicView. CreateEntity()
IExViewRepos. CreateEntity()

Finally the Adaptation logic module informs the Worker and Master agent with the *IExStatusReportView* interface.

3.1.3.2.3.2 Logic Update Required.

The Intelligent Adaptation Logic module suspends the operation of Worker agent and invokes the procedure of management logic module update from the Repository agent. Finally the IAL module informs the Worker and Master agent with the *IExStatusReportView* interface. The sequence diagram of this failure handling, is the same as in the case of Logic Unavailable failure (Figure 10) with the difference that in the field of the messages exchanged, the Logic update is requested instead of Logic creation.

3.1.3.2.4 Availability of Operational Resources

3.1.3.2.4.1 Critical Resource Utilization

This case is similar to the Resource unavailable failure with the difference that the Intelligent Adaptation Logic module after receiving a failure verification from Diagnostics agent, and if the failure persists, orders the release of some resources, or if fallback solution is followed, instructs Carrier module to migrate the Worker agent to a nearby alternative node (Figure 11). Finally, IAL module informs the Master and Worker agent. If the fallback solution is followed, IAL module periodically checks if recourses for a migration back to the previous network node (by using its Reattempt method) are available.

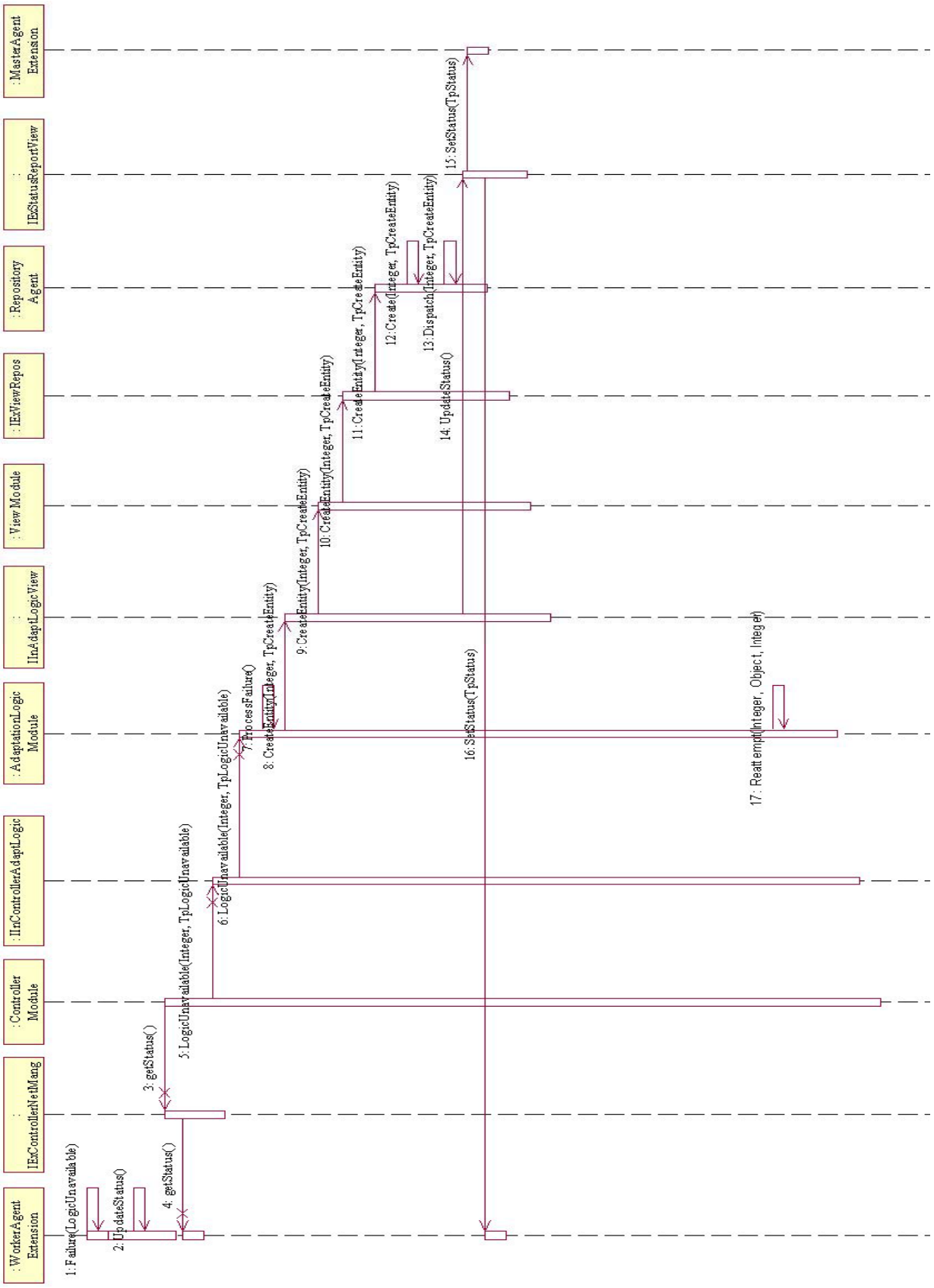


Figure 10 Logic unavailable sequence diagram

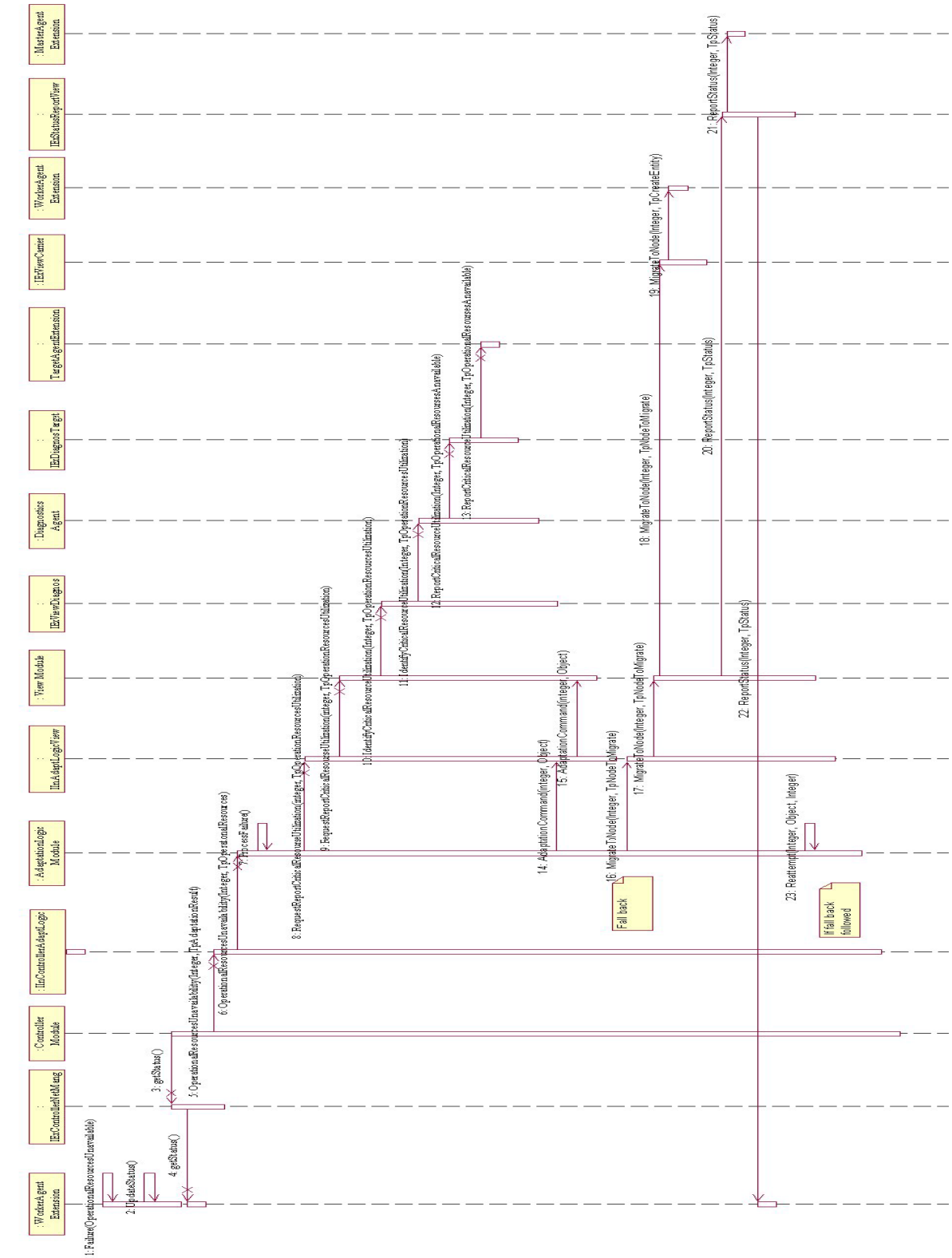


Figure 11 Unavailability of operational resources

4 Implementation

As it is aforementioned, this network performance monitoring system is based on Mobile Agent Technology (MAT) and specifically on the Grasshopper agent platform. In order for the system to be started some Grasshopper servers (one registry and three Grasshopper agencies) have to be created. In these Grasshopper servers some certain system entities have to be created and pre-exist before system's initiation. The Grasshopper servers and relevant system entities are:

Grasshopper server/ agency	**Role**	**Pre-existent agents**
reg	The registry of whole agent system	-
destination	The local place/agency where Worker performs its task	Target
destination	The place/agency where Worker migrates in order to perform its task	TargetWithRepositoryAnd Diagnostics Diagnostics Repository
alternative	The place/agency where Worker migrates, when a failure occurs.	Target
nms	The place where the managing entities are located.	Master Configurator UserAgent

The setup of the system starts with the creation of the registry, a destination Grasshopper agency to the targeted node, a destination and a nms Grasshopper agencies to the local node. In the following system description, it is considered that the system has two hosts, with IP addresses 169.254.212.10 and 169.254.154.26 respectively. The system after its setup has the following state and topology:

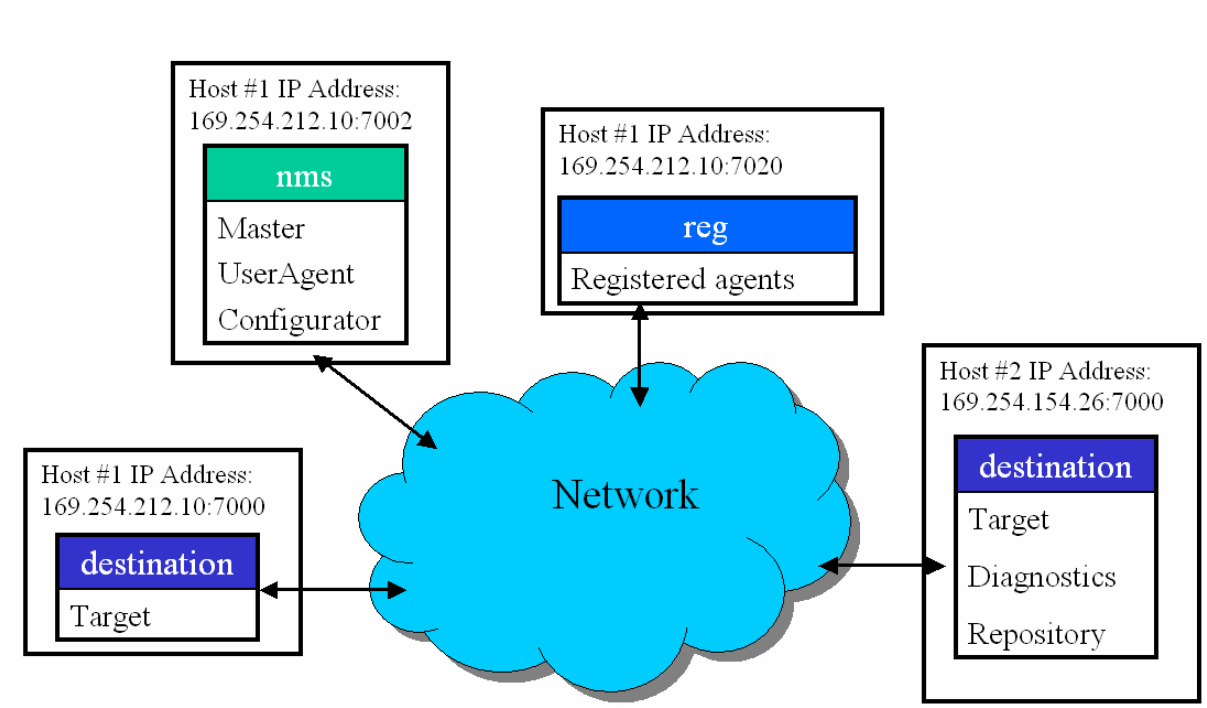


Figure 12 System's setup

When the User Agent is created in nms agency then, the system's graphical user interface appears. The standard Performance Monitoring System used for the initializing and controlling the system has the following interface

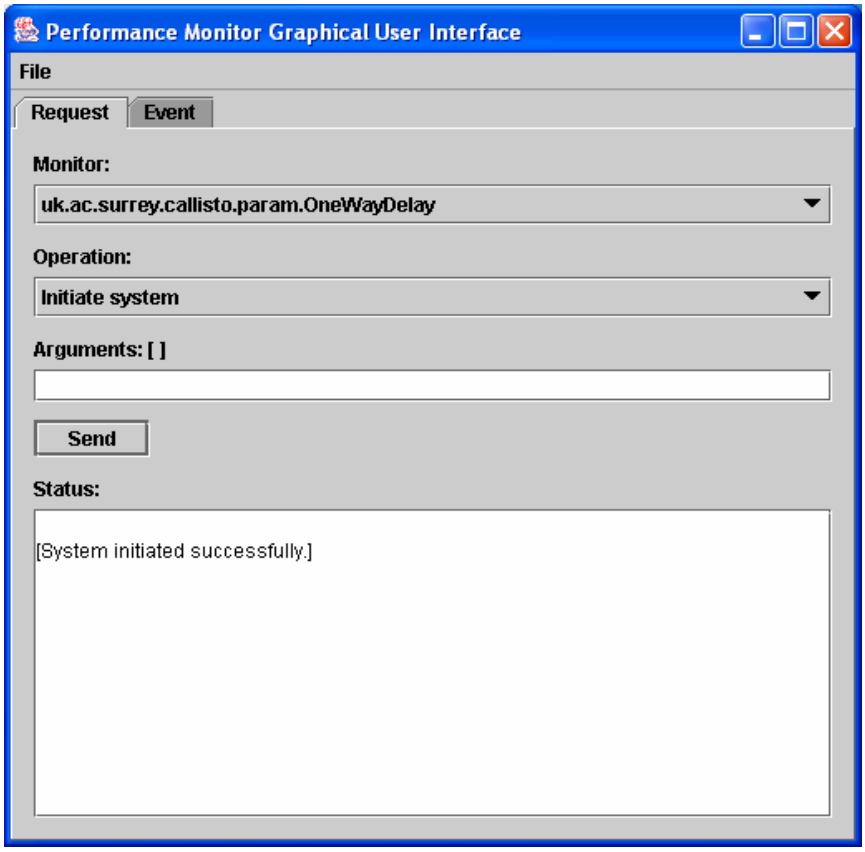


Figure 13 Performance Monitor Graphical User Interface.

From the above GUI, the user of the system, controls the performance monitoring system. More specifically, the user can select from the Operation panel to start and terminate the operation of the system, to change the report and granularity period and finally to add more thresholds or remove the already existing ones. Also, in the Monitor panel the user can select which of the two performance parameters (OneWayDelay or IPInputRate) wishes to be monitored.

In the following figure it can be shown which agents are registered in the same region registry when the performance monitoring system is running (after system setup).

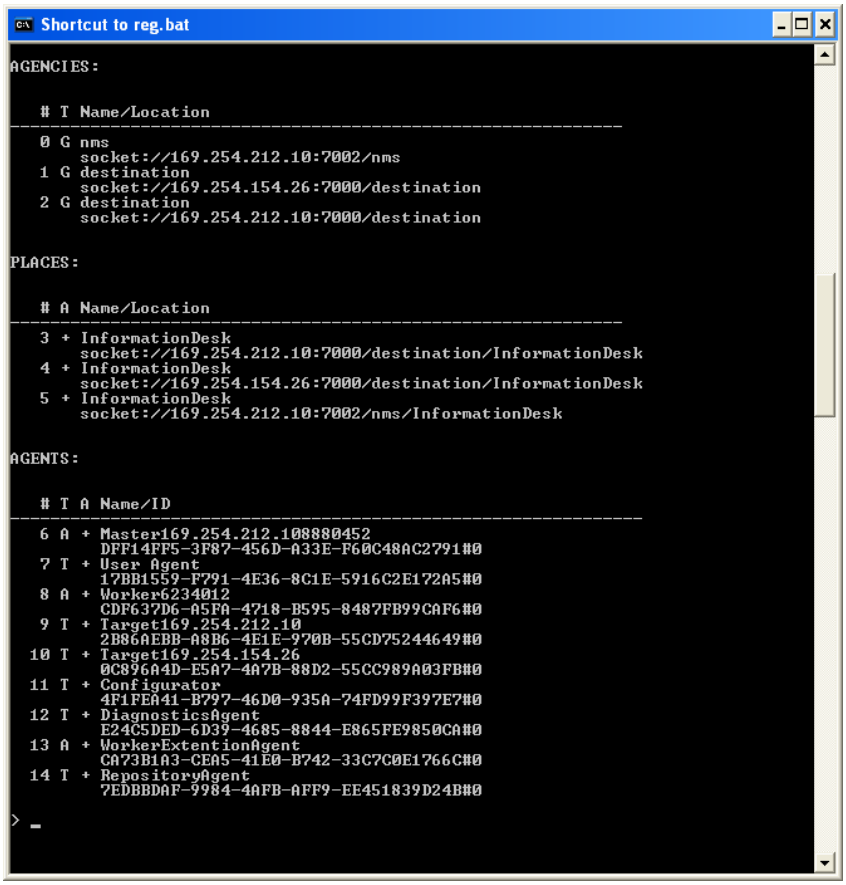


Figure 14 System's registry

After system's initiation, the first step is the creation of Location agent from Worker agent and subsequently its sending so as collecting the inputs. The LocationAgent migrates to both available hosts (destination places) and collects the inputs given from the Target agent that exist in each of the agencies and decides which of the two is the most suitable one for migration. It autonomously decides and reports to the Worker agent with the best destination loca-

tion. Finally, the LocationAgent is being removed from the Worker agent as it has finished its task.

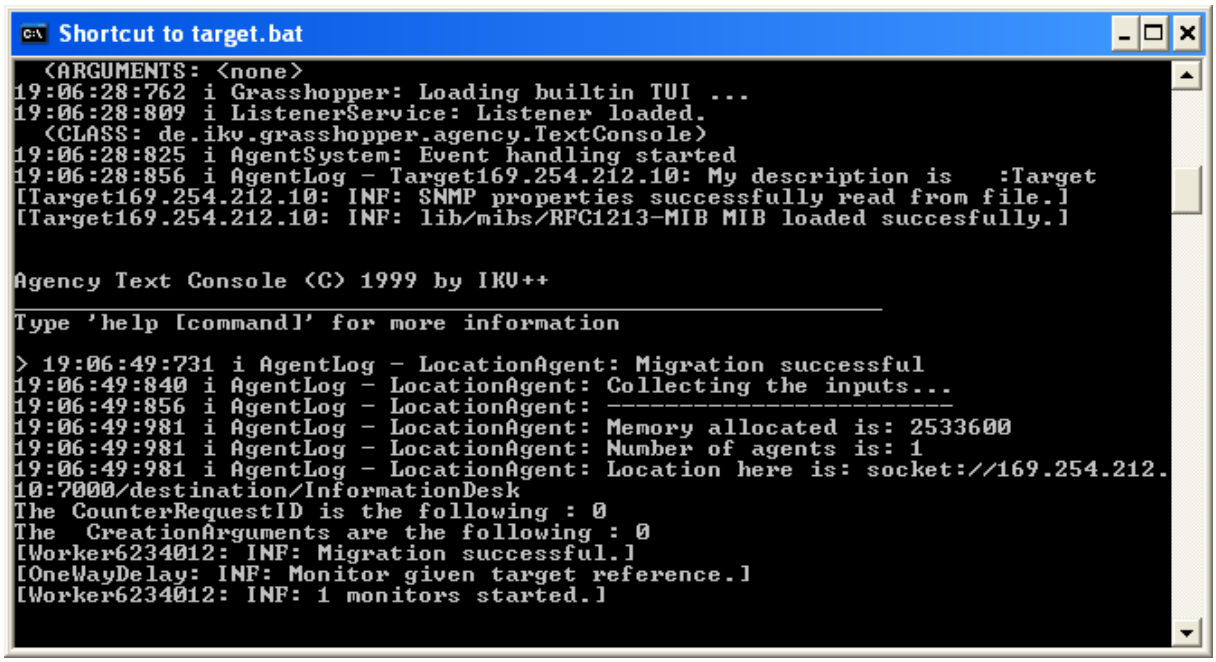


Figure 15 Location agent collects inputs from the first host (169.254.212.10)

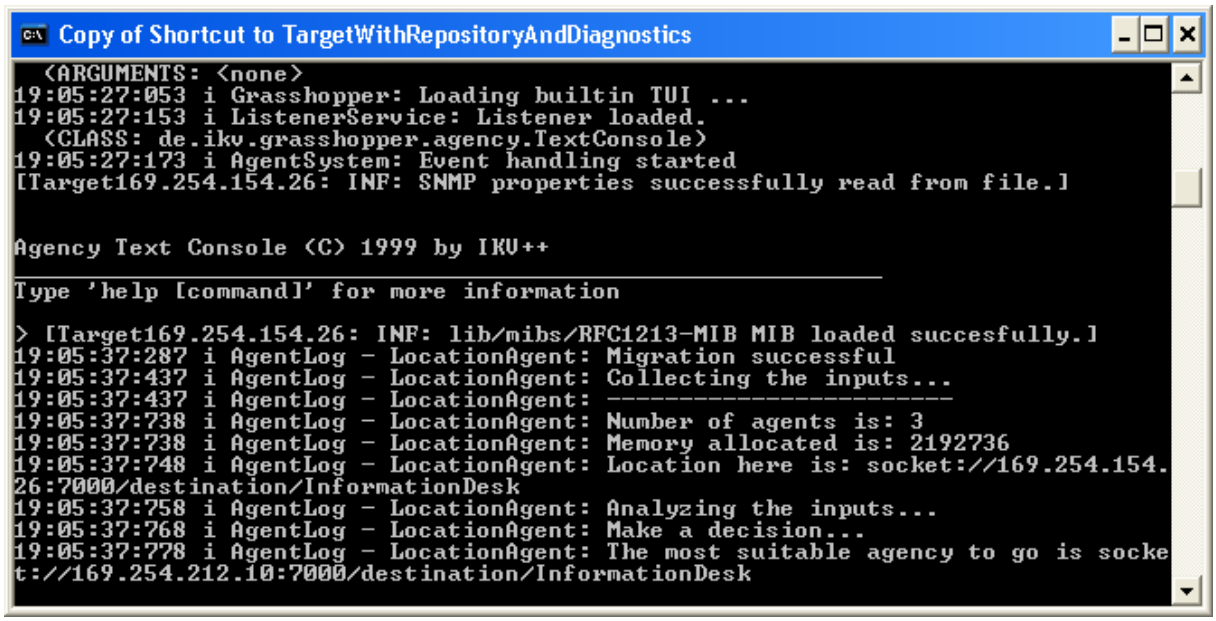


Figure 16 Location agent collects inputs from the second host (169.254.154.26)

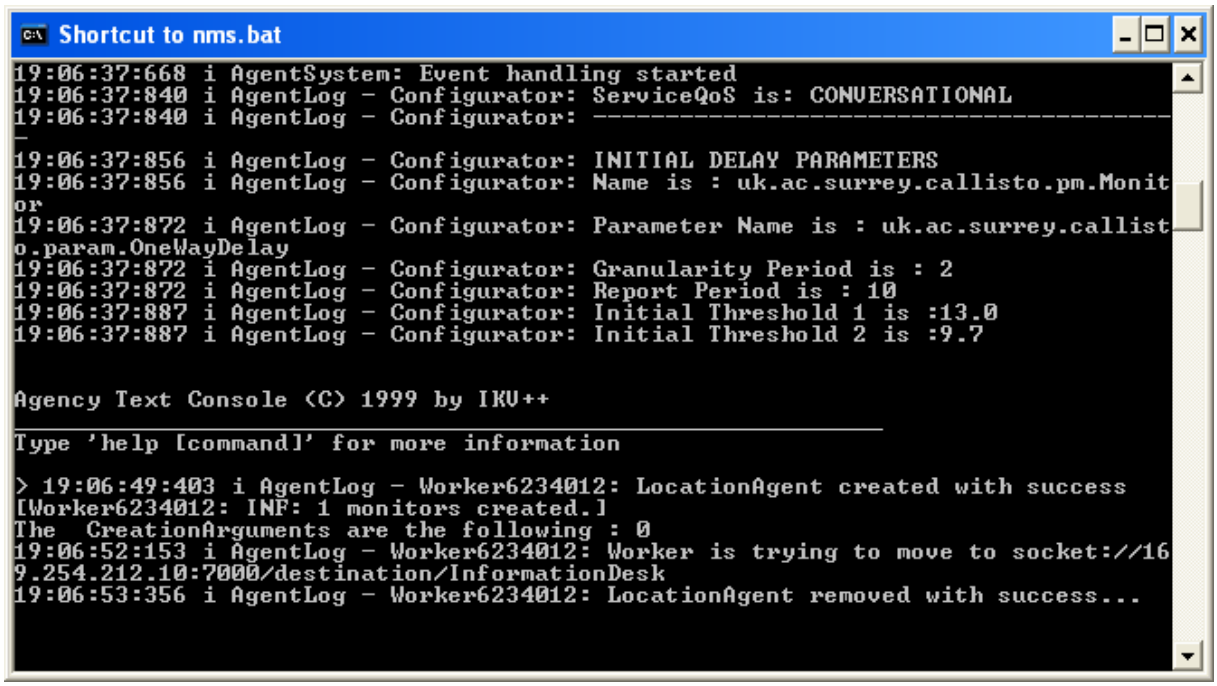


Figure 17 LocationAgent creation and removal after its task completion

The Worker agent is now ready to migrate to the chosen host and start monitoring in that node. The previous system contained the appropriate autonomy to help the user not to interfere with the system and automate many system operations.

After Worker's migration to the chosen host, Worker creates the WorkerExtentionAgent that handles the Intelligent Adaptation to the Environment Module (IAEM). The WorkerExtentionAgent provides a Graphical User Interface-GUI (Figure 18), by which the user can invoke several failures to the performance monitoring system and observe how the system responds to these. In this GUI can be chosen the failure category and the failure itself (see also paragraph 3.1.2.2.2 for failure description and strategy handling). There are six failure categories with their according sub-failures:

	Failure category	**Failure**
1	Availability of Correspondent	Correspondent unavailable Remote communication failure
2	Resource Availability	Resource unavailable
3	Availability of Collaborator	Collaborator unavailable
4	Availability of logic	Logic unavailable Logic update required
5	Proximity failures	Migration denied

		Runtime environment unavailable Network node inaccessible
6	Availability of operational resources	Critical resource utilization

Following is presented the Main Failures GUI in two different screen shots that depict the options, which can be selected. In this GUI the user may also provide the appropriate arguments to be considered by the Intelligent Adaptation to the Environment Module during failure handling.

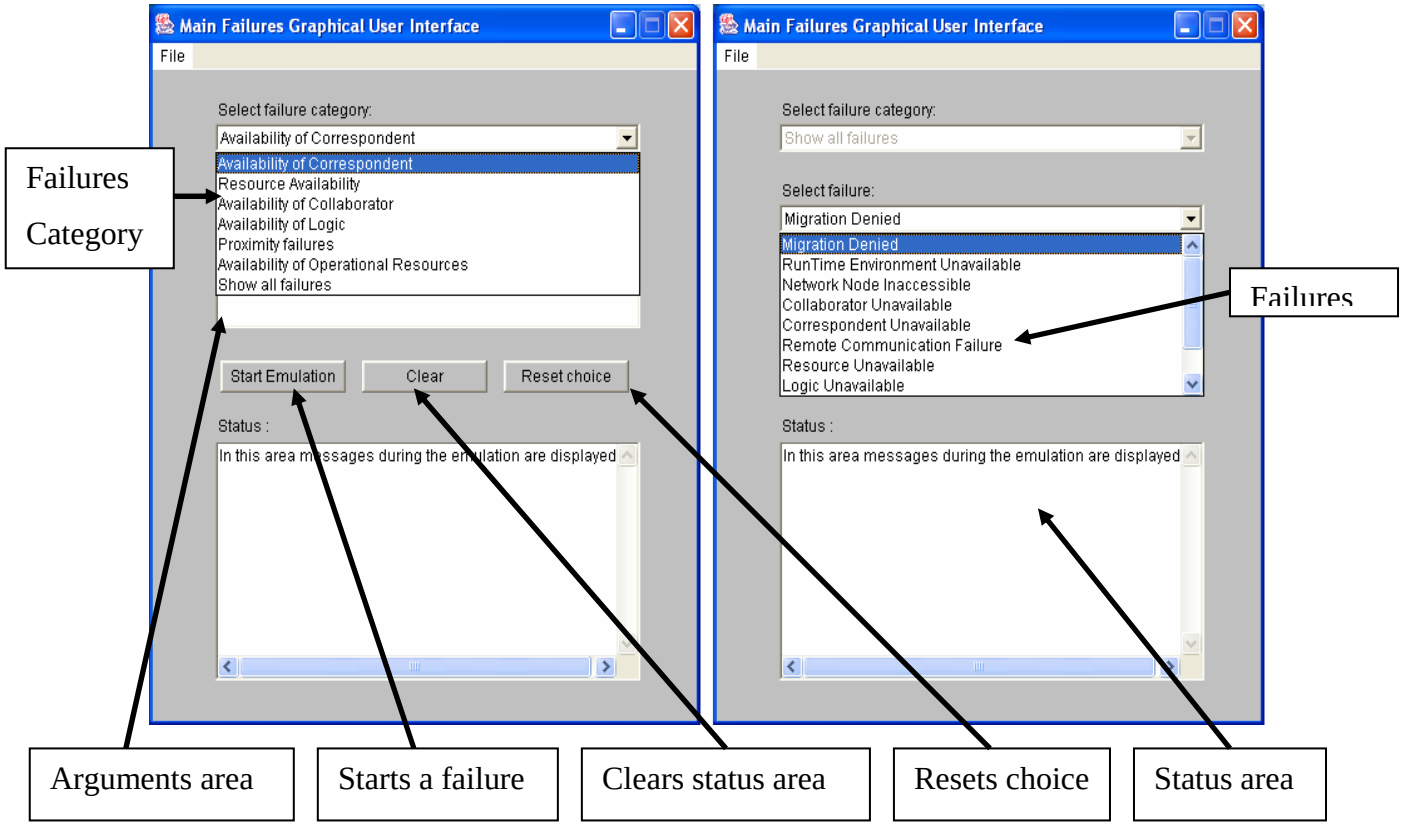


Figure 18 GUI for user interaction with the IAEM.

Additionally, the user by selecting File/Setup Environment is possible to set the IP addresses of the participating hosts in the performance monitoring system. This GUI has the following appearance (Figure 19)

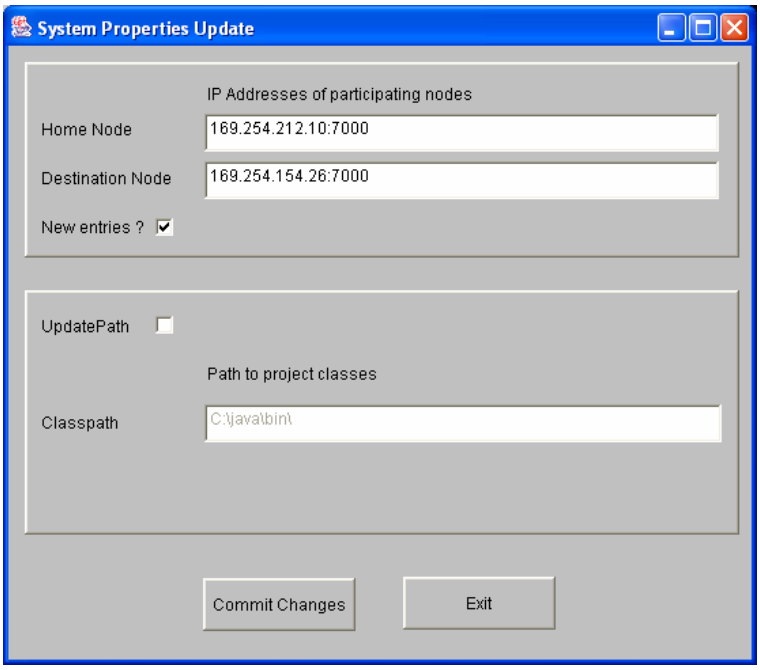


Figure 19 System's properties GUI.

The user may select from the Main Failures GUI (Figure 18) a failure or failures that wishes the system to handle. It should be noted that the Adaptable Fault-tolerant Performance Monitoring component is capable of handling multiple system failures and faults simultaneously and resolve each one separately. With this capability several different failures and faults may be invoked in the system e.g. Migration denied and resource unavailable failures and the system would handle them in the order of their invocation. Following is a short description of each fault handling from the system. It should be noted that the strategy of each fault handling has already been explained (see paragraph 3.1.3.2) and thus following are demonstrated some running scenarios.

Running Scenarios

Correspondent unavailable

This failure is initiated with the selection from the GUI of Figure 18 the Availability of Correspondent/ Correspondent unavailable choice. In order for the system to respond to this failure firstly, the Master agent (the Correspondent) is being removed from the Grasshopper place *nms*. After Master's agent removal, the system detects the failure of the Correspondent and initiates the procedure of Correspondent Unavailable as described in paragraph 3.1.3.2.2. Finally, for the sake of system's demonstration, the user is notified from Repository agent for Master's creation with the following GUI.

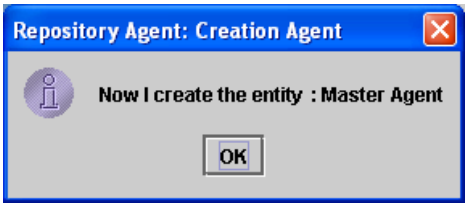


Figure 20 Master's Agent creation (Correspondent unavailable failure)

Resource unavailable

This failure is initiated with the selection from the GUI of Figure 18 the Resource Availability/ Resource unavailable choice. The system starts the procedure of diagnosing the failure by using the diagnostics agent (see also section 3.1.3.2.2). In case that a resource unavailability is diagnosed, the system informs the Master agent and periodically reattempts to find this resource. The notifications sent from the system to the Master agent, and system's reattempts to find this resource are shown in the following figure.

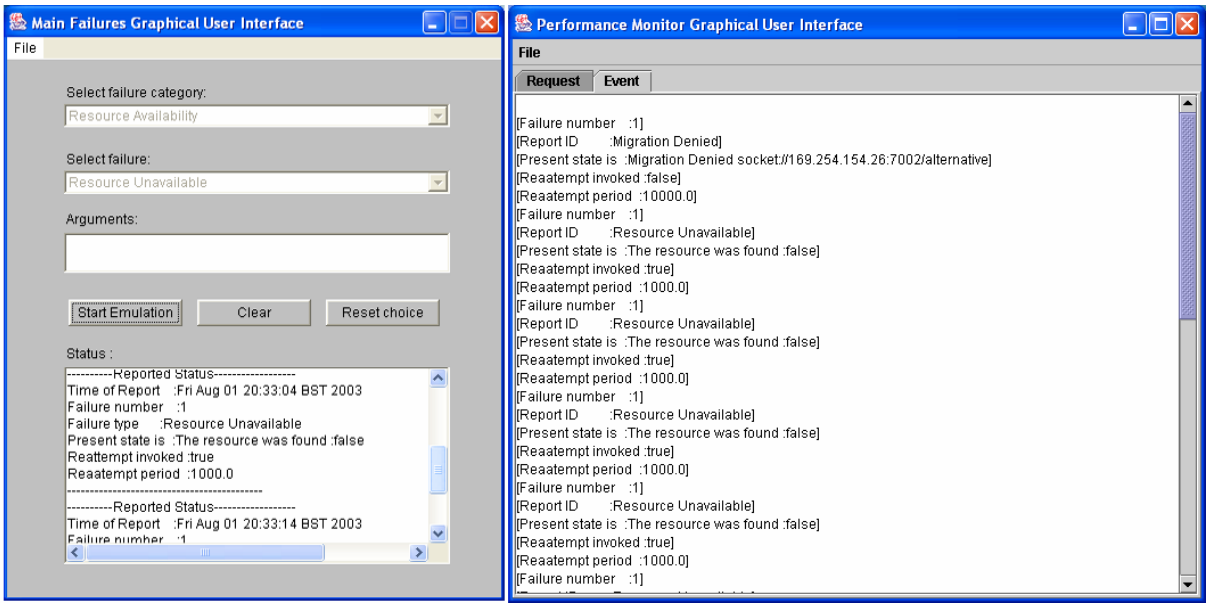


Figure 21 Resource unavailable failure.

Availability of Collaborator

This failure is invoked by selecting the Collaborator Availability /Correspondent unavailable from the GUI of Figure 18. The failure is dealt by the system and finally the Collaborator (Target agent) is created by the system in order for this failure to be resolved. The following figure shows the system condition during this failure handling.

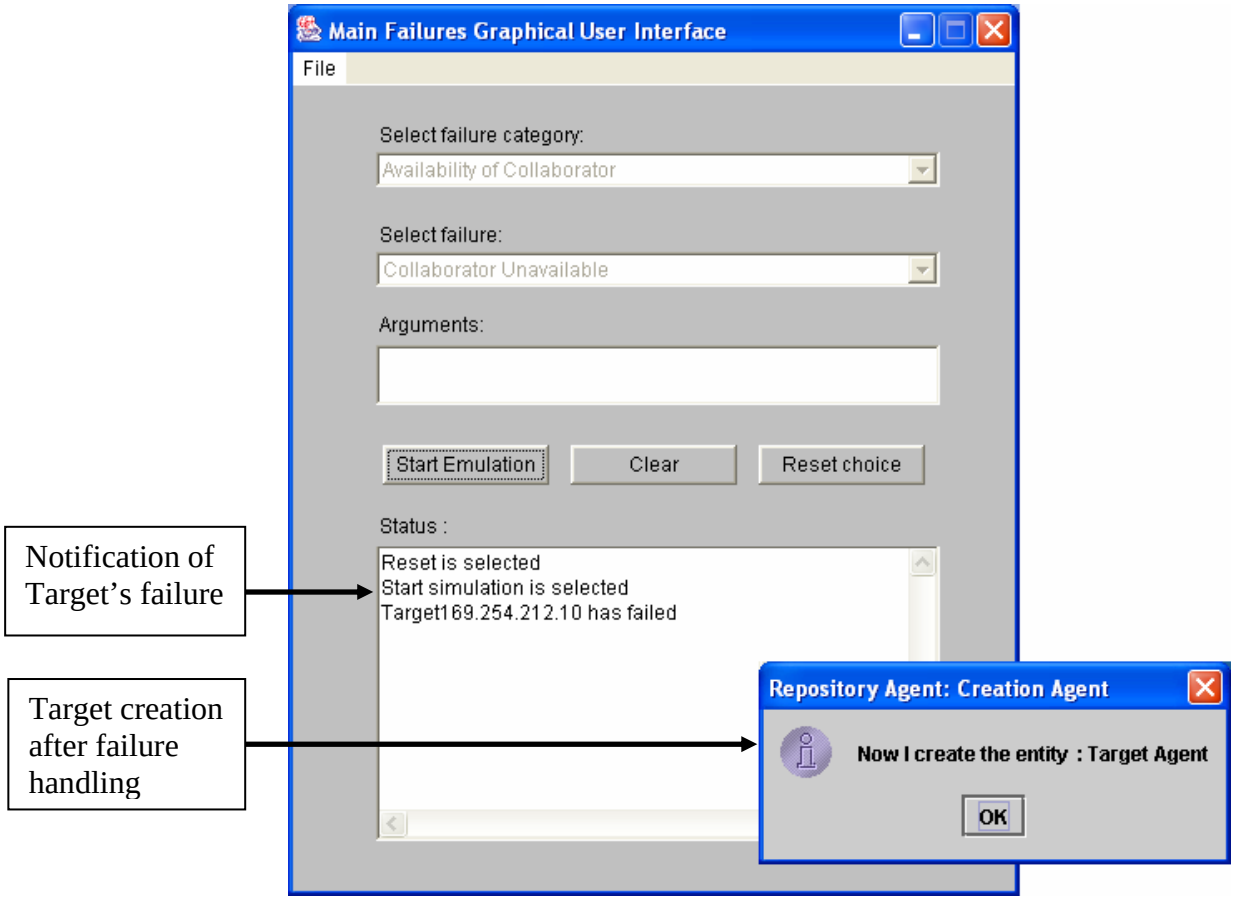


Figure 22 Collaborator unavailable failure

Availability of logic

Logic unavailability and update

Logic unavailability and logic update failures are invoked by selecting the Availability of Logic/ Logic Unavailable or Logic update required respectively from the GUI of Figure 18. These failures are dealt by the system and finally the appropriate logic is created by the system and dispatched to the requested node. The following figure shows the system condition during this failure handling.

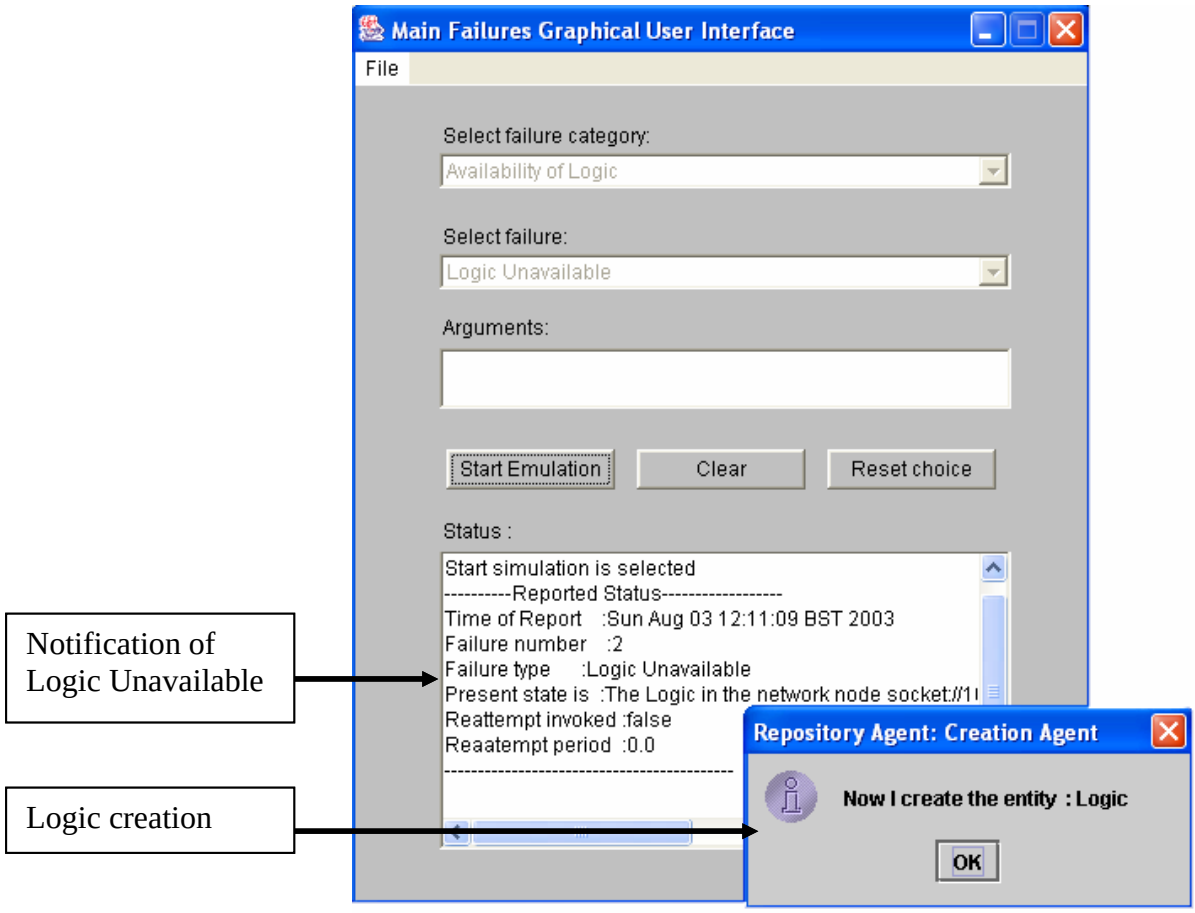


Figure 23 Logic unavailable failure handling

Proximity failures

This failure category includes three subcategories: Migration denied, Runtime environment unavailable and Network node inaccessible failures. Following is presented only the first failure handling (Migration denied) since from the one hand the handling of Runtime environment unavailable failure has the same strategy (and actions), and on the other hand the Network node inaccessible failure entails only a notification to be sent to the user and periodically check if the inaccessible node has been brought back to operational condition. The failure handling strategy of the system has been explained in paragraphs 3.1.3.2.1.1 and 3.1.3.2.1.2. The Migration Denied failure is invoked by selecting the Proximity failures / Migration Denied required from the GUI of Figure 18. This failure is dealt by the system and finally the Worker agent migrates to the network node that has been found from the adaptation procedure. The following figure shows the system condition during this failure handling.

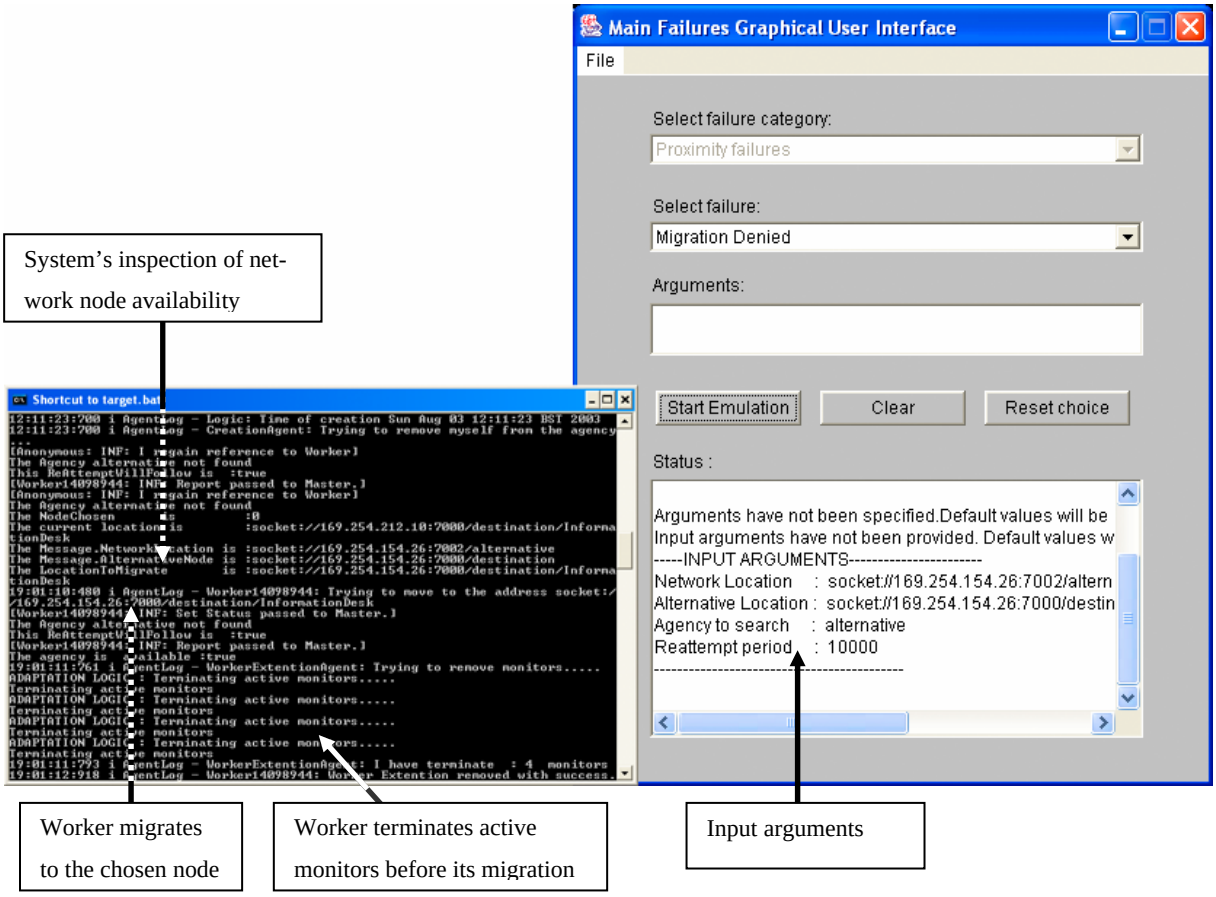


Figure 24 Migration denied failure

Availability of operational resources

Critical resource utilization

Critical resource utilization is invoked by selecting the Availability of operational resources/ Critical resource utilization failure from the GUI of Figure 18. This failure is dealt by the system (described in paragraph 3.1.3.2.4.1) and as a result, the Worker agent either releases some resources of the network node where is located or migrates to another alternative node. System's appearance during this failure is similar to the previous failure (Figure 24).

As previously mentioned the implementation of this project has been based on Mobile Agent Technology and more specifically on the Grasshopper Mobile agent platform version 2.4. The developed system is running on Windows XP Professional Edition and RedHat Linux v.8.4 operating systems. The developed system successfully tested on both operating systems and on PCs running different operating systems (one PC running Windows XP O/S and the other RedHat Linux O/S). The agents that have been implemented are all written in Java program-

ming language, which means that they can be used in any operating system that supports Java Virtual Machine.

Finally, it has also been developed the API (Application Programming Interface) of this project, and in the following figure it can be seen a screen-shot of this API (Figure 25). This API has been developed with the tools provided by the J2SDK1.40 (Java Standard Development Toolkit version 1.4).

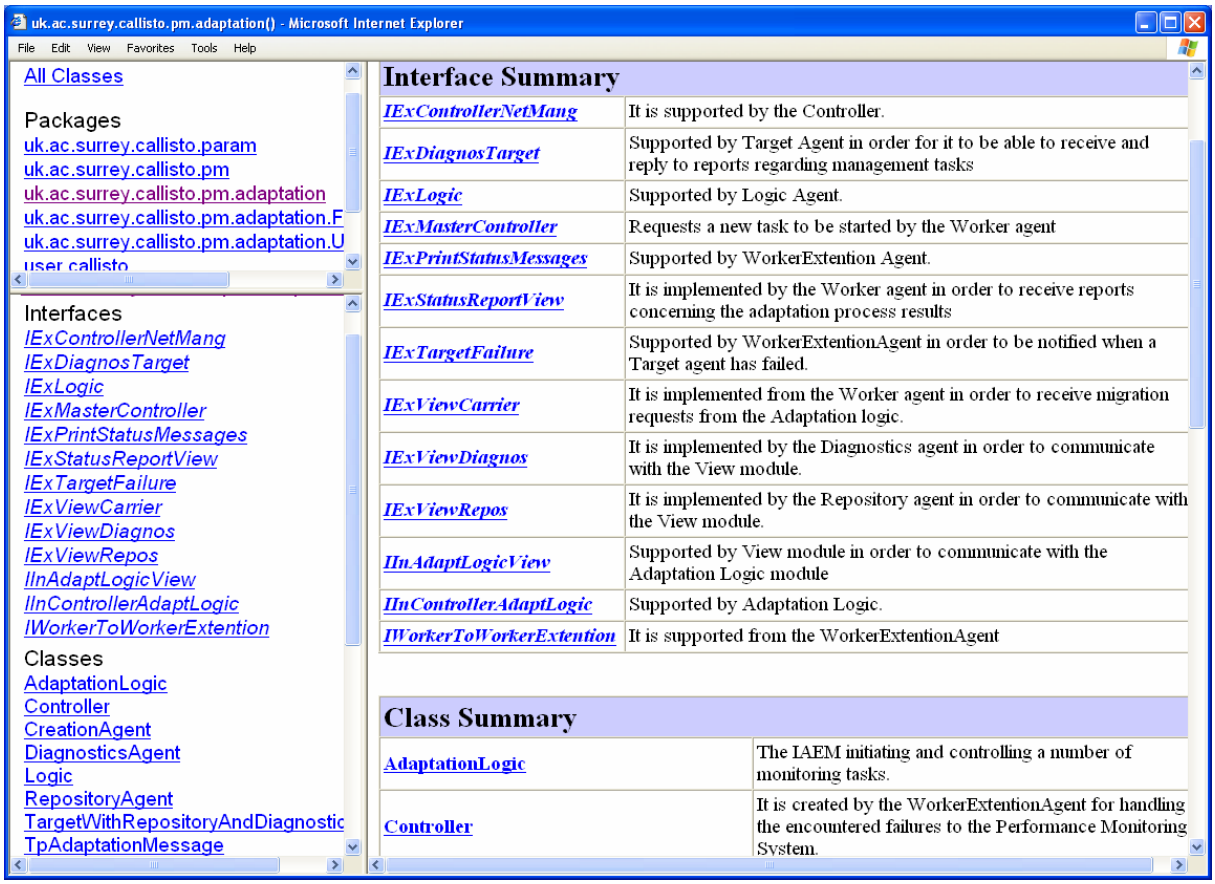


Figure 25 System's API

5 Achievements

The aim of this project has been the enhancement of a network performance monitoring system based on mobile agents, with the intelligence, flexibility and autonomy in order to handle several failures that might be encountered during network performance motoring operation. Failures and errors are dealt by the system, without user's interaction or involvement. The existent network performance monitoring has been enhanced with functionality similar to that of the fault management. The developed network performance monitoring system contains the intelligence that allows the autonomous fault management of the performance monitoring components. The system is able to handle communication failures such as, availability of

network monitoring entities, required updates of management functionality, resource over-utilization, and agent migration issues.

The developed system is a reusable adaptation to the environment component that can be used in any performance monitoring system implemented with Mobile Agent Technology, without important modifications to that system. Moreover, this system allows a more autonomous Performance Monitoring solution since it significantly reduces administration intervention to restore system faults and failures. Furthermore, the developed components help in the direction of ensuring the correct operation of the Performance Monitoring system since it provides the necessary functionality of bringing the system back and keeping it, to a stable condition after a fault or failure occurrence. Thus, though not only the correct operation and functionality of a Performance Monitoring system is ensured but also its stability against potential faults and failures of this system as a whole.

In summary, the components developed in this project have the following characteristics:

- Handle communication failures.
- Handle availability of network monitoring entities.
- Handle updates of management functionality.
- Handle conditions of resource over-utilization and agent migration issues.
- Are reusable adaptation to the environment components.
- Allow a more autonomous Performance Monitoring solution through less administration intervention to restore system faults and failures.
- Help in the direction of ensuring the correct and stable operation of the Performance Monitoring system.

6 Conclusions and Future Directions

The present project has been focused on the enhancement of a network performance monitoring system with the required intelligence, flexibility and autonomy to handle some specific failures that might be encountered during network performance motoring operation. The resulting network performance system is stable, autonomous, and necessitates less administration involvement during its operation.

This system implements the Mobile Agent Technology (MAT) on the network management functions and more specifically on network performance monitoring. MAT is a promising technology that although is still on the realm of research, has many areas of applicability besides network management. Work and proposals of MAT on 3G networks service modelling

creation, manipulation and provision [9-12] and also in providing network security functions [13].

Coming back to the network performance monitoring area, and in the current project, there is some additional functionality that could be implemented in this network performance monitoring system but in the tight time constrains of a MSc project couldn't be addressed and even more be implemented. Below some thoughts about what could be implemented in such a system in future MSc projects, are mentioned:

- Communication security among the collaborating entities.
- Dynamic discovery of other managing entities and dynamic collaboration with them. (System openness and scalability)
- Dynamic discovery of other adaptable fault tolerant components and dynamic collaboration with them.
- Provision for whole system's breakdown and effective recovery.
- Provision for more performance parameters observation, correlation and diagnostic testing.
- Processing of the collected information for analyzing the system failures and faults so as to prevent these to happen in the future.
- Isolation of network management resources that are insecure or are prone to failures.

This project gave to the author of this MSc thesis the opportunity to learn in depth the functionalities of a network management system and control in conjunction with the exploitation of Mobile Agent Technology (MAT) as well as its implementation in such a system. The developed network performance monitoring system exhibits autonomous behaviour and adaptable fault tolerance. The Intelligent Adaptation to the Environment component that was perceived, analyzed designed and implemented in the framework of this MSc project, could be integrated with any other network performance monitoring system without significant effort and major changes.

7 References

[1] James Curtis, "Analysis of Voice Over IP Traffic", October 6, 1999 Computer Based Learning Unit, University of Leeds. Available at http://wand.cs.waikato.ac.nz/wand/ publications/jamie_420 /final/report.html.

[2] ITU-T Rec. X.739, Information Technology - Open Systems Interconnection, Systems Management Functions - Metric Objects and Attributes, 1992.

[3] Grasshopper Agent Platform and Basic Concepts, http://www.ikv.de/products/ grasshopper/ index.html

[4]W.Stallings SNMP, SNMPv2, SNMPv3 and RMON 1 and 2 Addison Wesley Third Edition 1999

[5]Danny B.Lange /Mitsuru Osima " Programming and Deploying Java Mobile Agents with Aglets" Addison Wesley 1998.

[6] M. Cheikhrouhou, P. Conti, R. Oliveira, J.Labetoulle, *Intelligent Agents in Network Management A state-of-the-art*, 1997, http://www.eurecom.fr/~diana/ publications/ September 97 /soa-toc.html

[7]G. Pavlou Lecture notes from the MSc module "Network & Service Management and Control". University of Surrey.

[8] C. Bohoris, A. Liotta, G. Pavlou "Software Agent Constrained Mobility for Network Performance Monitoring" University of Surrey.

[9] Vasileios A. Baousis "Development of Virtual home environment (VHE) and location dependent services through mobile agent technology" MSc Thesis, University of Athens, Department of Informatics and Telecommunications, Greece.

[10] Markus Breust, Lars Hagen, Thomas Magedanz, 'Impacts of Mobile Agent Technology on Communications System Evolution', IKV++ GmbH Technical University of Berlin, From IEEE 8/1998

[11]Thomas Magedanz, K. Rothermel, S. Krause, 'Intelligent Agents: An Emerging Technology for Next Generation Telecommunications?' INFOCOM 3/96

[12] “UMTS Network Aspects Service Scenario for the VHE concept”, EURESCOM, ANNEX PIR.21

[13] Wayne A. Jansen “ Intrusion detection with mobile agents” Computer Communications 25(2002) pages 1392-1401

8 Appendix: Source code.

1.Worker agent

```
package uk.ac.surrey.callisto.pm;
import uk.ac.surrey.callisto.pm.adaptation.*;
import uk.ac.surrey.callisto.pm.adaptation.FailuresGUI.*;
import uk.ac.surrey.callisto.pm.adaptation.UpdateProperties.*;
import de.ikv.grasshopper.agent.*;
import de.ikv.grasshopper.agency.*;
import de.ikv.grasshopper.type.*;
import de.ikv.grasshopper.communication.*;
import de.ikv.grasshopper.util.*;
import javax.swing.JOptionPane;
import java.util.*;
import java.net.*;
import java.io.*;
/** The worker initiating and controlling a number of monitoring tasks.  */
public class Worker extends SmartAgent implements
WorkerRequest,MonitorEvent,SelectedDestination, IExStatusReportView,IExViewCarrier,
Serializable  {
    /** The associated target used for active measurements. */
      private TargetRequest activeTargetRef = null;
    /** The local target. */
      private TargetRequest passiveTargetRef = null;
    /** A reference to the LocationAgent. */
     private AvailableLocations locationAgentRef;
    /** The Id of the LocationAgent*/
     private Identifier locationAgentId;
    /** Migration to the destination location. */
     private String destinationLocation;
    /** Keep all performance monitoring tasks. */
    private TpTaskList taskList;
    /** The master used. */
    private WorkerEvent masterRef = null;
    /** The state of the agent life-cycle. */
    private int state;
    /* The executing monitors. */
   private MonitorRequest[] monitorList;
    /** Designates a termination state. */
    private boolean terminate;
    /** Logging facilities. */
    private Log log;
    /** The object name. */
    private String name = null;
    private Vector s = new Vector();
    AgentInfo mobileAgentInfo = null;
    AgentInfo WorkerTaskManagerInfo=null;
```

```
    Migrator MigratorManager;
    transient IRegion RegionProxy = null;
    transient IRegion regionProxy =null;
    transient AgentInfo  WorkerExtentionInfo = null;
    transient Finalizer CheckWorker=null;
    transient IAgentSystem agencyProxy= null;
    transient protected Object waitLock;
    transient TpResult LiveResult = null;
    transient IRegion WholeRegionProxy= null;
    transient boolean migrate;
    private Object [] CreationArguments = new Object[1];
    private int counterMigrations=0;
    private int failures=0;
//-------------------------------------------------------------------------//
    /** Empty object initialization.  */
public Worker() {}
//-------------------------------------------------------------------------//
    /** This method is used to read the IP addresses of participating servers
     * into the Monitoring System. This information is contained into the
     * file  ParticipatingServers.properties which is located on the path
     * C:/java/bin/lib/. The file ParticipatingServers.properties is updated
     * from the main GUI "Environment setup".
     * @return .An array of Strings containing the IP of participating
     * servers.
     */
public  String [] readIPAddressesOfAvailableLocation (){
     ReadWrite ReadFromFile = new ReadWrite();
     Vector  Servers = ReadFromFile.read("./lib/ParticipatingServers.properties");
     String [] ParticipatingServers= new String [Servers.size()];
                      try  {
           for( int index=0; index <(Servers.size());index++){
ParticipatingServers[index]="socket://"+(String)Servers.elementAt(index)+":7000/destination
";
              // log("Participating servers are :  "+ ParticipatingServers[index]);
           }
       }
       catch (Exception writeErr){
         writeErr.printStackTrace();
                 }
     ReadFromFile=null;
return ParticipatingServers;
 }
//-------------------------------------------------------------------------//
    /**
     * Provides the agent name.
     *
     * @return The name of the Worker agent.
     */
public String getName() {
       name = "Worker" + hashCode();
       return name;
```

```
    }
//-------------------------------------------------------------------------//
    /**
     * Provides an agent description.
     * @return The description.
     */
    public String getDescription() {
        return "Worker";
    }
//-------------------------------------------------------------------------//

public void beforeRemove(){
   if (WorkerExtentionInfo != null)  {
     TpResult ReturnValue =terminate();
     RegionProxy = getRegion();
     AgentInfo [] SearchForWorkerExtention;
         SearchFilter filter = new
SearchFilter(SearchFilter.DESCRIPTION+"=WorkerExtentionAgent");
         SearchForWorkerExtention = RegionProxy.listAgents(null,filter);
         agencyProxy = getAgentSystem();
    try {
        agencyProxy.removeAgent(SearchForWorkerExtention[0].getIdentifier());
        log("Worker Extention removed with success...");
        WorkerExtentionInfo = null;
        }
   catch(Exception e){
    log("Removing WorkerExtentionAgent failed..");
    e.printStackTrace();
}}}
//-------------------------------------------------------------------------//
public void beforeMove(){
    if (WorkerExtentionInfo != null)  {
        RegionProxy = getRegion();
        AgentInfo [] SearchForWorkerExtention;
            SearchFilter filter = new
SearchFilter(SearchFilter.DESCRIPTION+"=WorkerExtentionAgent");
            SearchForWorkerExtention = RegionProxy.listAgents(null,filter);
            agencyProxy = getAgentSystem();
        try {
            agencyProxy.removeAgent(SearchForWorkerExtention[0].getIdentifier());
            log("Worker Extention removed with success...");
            WorkerExtentionInfo = null;
            }
    catch(Exception e) {
        log("Removing WorkerExtentionAgent failed..");
        e.printStackTrace();
    }    } }
//-------------------------------------------------------------------------//
    /**
     * This method is called after the migration of Worker to another network node.
     */
```

```
public void afterMove (){
agencyProxy = getAgentSystem();
       try {
            WorkerExtentionInfo =
agencyProxy.createAgent("uk.ac.surrey.callisto.pm.adaptation.WorkerExtentionAgent",getInf
o().getCodebase(),null,CreationArguments);
       } catch (AgentCreationFailedException e) {
          log("Failed to create WorkerExtentionAgent.");
       }
 if  Reattempt==true&&RequestID!=0&&TpMessage!=null&&MessageType!=null){
       log("Starting reaatempt in WorkerExtention Agent.");
            (new Reaatempt()).start();
 } }
//-------------------------------------------------------------------------//
public void action(){
   if (WorkerExtentionInfo!=null){
   agencyProxy = getAgentSystem();
       try {
          WorkerExtentionInfo =
agencyProxy.createAgent("uk.ac.surrey.callisto.pm.adaptation.WorkerExtentionAgent",getInf
o().getCodebase(), null, CreationArguments);
       } catch (AgentCreationFailedException e) {
          log("Failed to create WorkerExtentionAgent.");
        }}}
//-------------------------------------------------------------------------//
   /**
    * Specify agent behaviour.
    */
public void live() {
   int yesOrNo= 0;
   counterMigrations++;
   String [] ParticipatingServersRead = readIPAddressesOfAvailableLocation();
   IExViewDiagnos ReferenceToDiagnostics = getReferenceToDiagnosticsAgent();
   IExViewRepos ReferenceToRepository = getReferenceToRepositoryAgent();
   CreationArguments[0]= new Integer(failures);
   System.out.println("The  CreationArguments are the following : "+ failures);
  switch(state) {
       case 0:
          break;
       case 1:
                log.print(Constants.INFO, "Migration successful.");
                // Get host IP address and lookup for target.
                InetAddress localHost = null;
                try {
                   localHost = InetAddress.getLocalHost();
                } catch (Exception unknownHost) {
                   log.print(Constants.ERROR, "Unknown local host.");
                   log.print(Constants.DEBUG, unknownHost.toString());
                }
                // Initialize monitors.
                int j;
```

```
                TpResult result = null;
                for (j = 0; j < monitorList.length; j++) {
                   if (monitorList[j].getMeasureType() ==
PerformanceParameter.PASSIVE_MEASURE) {
                      if (passiveTargetRef == null) {
                         try {
                            passiveTargetRef =
lookupTargetAgent((InetAddress.getLocalHost()).getHostAddress());
                         } catch(Exception e) {
                            log.print(Constants.ERROR, "Error while retrieving localhost
address.");
                            log.print(Constants.DEBUG, e.toString());
                         }
                      }
                      result = monitorList[j].setTarget(passiveTargetRef);
                   } else if (monitorList[j].getMeasureType() ==
PerformanceParameter.ACTIVE_MEASURE) {
                      if (activeTargetRef == null)
                         activeTargetRef = lookupTargetAgent(taskList.targetNode);
                      result = monitorList[j].setTarget(activeTargetRef);
                   }
                }
                log.print(Constants.INFO, (String)(j + " monitors started."));
             yesOrNo = JOptionPane.showConfirmDialog(null,"Start monitors?\n[Hint:If yes,
migration of Worker agent will not be feasible later
on]","Worker",JOptionPane.YES_NO_OPTION);
             if (yesOrNo == JOptionPane.YES_OPTION) {
                System.out.println("Starting monitors.....................................................");
                for (int index = 0; index < monitorList.length; index++){
                   result = monitorList[index].startMonitoring(getRegion());
                }             }
                // Initialize the finalizer.
                (new Finalizer()).start();
                // Initialize the Migrator.
                (new Migrator()).start();
                      break;
       }    }
//-------------------------------------------------------------------------//
   /*
    * Look up for a Target agent in the specified host. Returns the Target agent proxy or null if
no Target agent was found.
    */
   private final TargetRequest lookupTargetAgent(String targetHost) {
      AgentInfo[] targetInfo;
      regionProxy = getRegion();
      TargetRequest targetProxy = null;
      SearchFilter filter = new SearchFilter(SearchFilter.NAME + "=Target" + targetHost +
"&" + SearchFilter.LOCATION + "~" + targetHost);
      targetInfo = regionProxy.listAgents(null, filter);
      if (targetInfo.length != 0) {
```

```
            targetProxy = (TargetRequest) ProxyGenerator.newInstance(TargetRequest.class,
targetInfo[0].getIdentifier().toString(), ProxyGenerator.SYNC);
        } else {
            log.print(Constants.ERROR, "Cannot locate Target.");
        }
        if (targetProxy == null) {
            log.print(Constants.ERROR, "No target.");
        }
        return targetProxy;    }
//-------------------------------------------------------------------------//
    /**
     * Agent initialization.
     * @param The initial arguments.
     */
    public void init(Object[] args) {
        // Enable logging.
        log = new Log();
        log.enabled = true;
        log.name = name;
        state = 0;     // Agent is at home agency.
        Identifier masterId = (Identifier) args[0];
        masterRef = (WorkerEvent)ProxyGenerator.newInstance(WorkerEvent.class, masterId);
        taskList = new TpTaskList(((TpTaskList)args[1]).monitorList,
((TpTaskList)args[1]).targetNode);
        terminate = false;
        migrate= false;
        Object[] argsnew = new Object[readIPAddressesOfAvailableLocation().length];
            for(int j=0;j<readIPAddressesOfAvailableLocation().length;j++) {
            s.addElement(readIPAddressesOfAvailableLocation()[j]);                }
       AgentInfo mobileAgentInfo = null;
    //Get proxy of local agency
    agencyProxy = getAgentSystem();
    //Create LocationAgent in order to collect the inputs
        try {
            for(int i=0;i<s.size();i++) {
            argsnew[i] = s.get(i);
            }
        mobileAgentInfo =
agencyProxy.createAgent("uk.ac.surrey.callisto.pm.LocationAgent",getInfo().getCodebase(),
null,argsnew);
        log("LocationAgent created with success ");        }
        catch (AgentCreationFailedException e)   {
        log("Failed to create LocationAgent ");
        e.printStackTrace();        }
    //Create a proxy in order to communicate with the LocationAgent
    if(mobileAgentInfo != null)
    locationAgentRef =
(AvailableLocations)ProxyGenerator.newInstance(AvailableLocations.class,mobileAgentInfo
.getIdentifier());
    locationAgentId = mobileAgentInfo.getIdentifier();
        int size = (taskList.monitorList).length;
```

```
        monitorList = new MonitorRequest[size];
        ArrayList monitorArray = new ArrayList();
        monitorArray = (ArrayList)args[2];
        // Create monitors.
        int i;
            for (i = 0; i < size; i++) {
                monitorList[i] = (MonitorRequest) monitorArray.get(i);          }
        log.print(Constants.INFO, (String)(i +" monitors created."));    }
//-------------------------------------------------------------------------//
    /**
     * Add a threshold to a monitored performance parameter.
     *
     * @param parameterName The name of the monitored performance parameter.
     * @param threshold The new threshold.
     * @return The result of the method call.
     */
    public TpResult addThreshold(String parameterName, TpThreshold threshold) {
        int index = lookupMonitorIndex(parameterName);
        TpResult result = monitorList[index].addThreshold(threshold);
        return result;
    }
//-------------------------------------------------------------------------//
    /**
     * Remove a threshold of a monitored performance parameter.
     * @param parameterName The name of the monitored performance parameter.
     * @param thresholdLevel The threshold level to remove.
     * @return The result of the method call.
     */
    public TpResult removeThreshold(String parameterName, double thresholdLevel) {
        int index = lookupMonitorIndex(parameterName);
        TpResult result = monitorList[index].removeThreshold(thresholdLevel);
        return result;
    }
//-------------------------------------------------------------------------//
    /**
     * Change the granularity period of a monitored performance parameter.
     * This corresponds to the period within which two successive measurements
     * are taken from the targeted network node.
     * @param monitorName The name of the monitored performance parameter.
     * @param timePeriod The new granularity period in seconds.
     * @return The result of the method call.
     */
    public final TpResult changeGranularityPeriod(String parameterName, long timePeriod) {
        int i = lookupMonitorIndex(parameterName);
        TpResult result = monitorList[i].changeGranularityPeriod(timePeriod);
        return result;
    }
//-------------------------------------------------------------------------//
    /**
     * Change the report period of a monitored performance parameter.
     * This corresponds to the period at the end of which a report is sent
```

```
     * containing performance information taken from the targeted network node.
     * @param parameterName The name of the monitored performance parameter.
     * @param timePeriod The new report period in seconds.
     * @return The result of the method call.
     */
    public TpResult changeReportPeriod(String parameterName, long timePeriod) {
        int i = lookupMonitorIndex(parameterName);
        TpResult result = monitorList[i].changeReportPeriod(timePeriod);
        return result;
    }
//-------------------------------------------------------------------------//
    /**
     * Suspend the operation an existing monitor.
     * @param parameterName The name ofthe monitor to suspend.
     * @return The result of the method call.
     */
    public TpResult suspendMonitor(String parameterName)    {
        int i = lookupMonitorIndex(parameterName);
        TpResult result = monitorList[i].suspend();
        return result;    }
//-------------------------------------------------------------------------//
    /**
     * Resume the operation of a suspended monitor.
     * @param parameterName The name ofthe monitor to suspend.
     * @return The result of the method call.
     */
    public TpResult resumeMonitor(String parameterName) {
        int i = lookupMonitorIndex(parameterName);
        TpResult result = monitorList[i].resume();
        return result;    }
//-------------------------------------------------------------------------//
    /**
     * Terminate the operation of the QoS management system.
     * @return The result of the method call.
     */
    public TpResult terminate() {
        int i;
        for (i = 0; i < monitorList.length; i++) {
            monitorList[i].terminate();        }
        log.print(Constants.INFO, (String)("removed " + i + " monitors."));
        terminate = true;
        return (new TpResult(Constants.P_RESULT_SUCCESS));    }
    /* END: WorkerEvent Interface */
    /* BEGIN: MonitorEvent Interface */
//-------------------------------------------------------------------------//
public void terminate (boolean TerminateDecision){
    terminate = true;
}
//-------------------------------------------------------------------------//
private boolean getTerminateStatus(){
 return terminate; }
```

```
/* END: WorkerEvent Interface */
/* BEGIN: MonitorEvent Interface */
//---------------------------------------------------------------------------//
    /**
     * Provides a report of performance information.
     * @param report A report containing performance information.
     * @return The result of the method call.
     */
    public TpResult report(TpReport report) {
        TpResult result = masterRef.report(report);
        log.print(Constants.INFO, "Report passed to Master.");
        return result;    }
//---------------------------------------------------------------------------//
    /**
     * Provides a performance notification based on the checking of thresholds.
     * @param notification A performance notification.
     * @return The result of the method call.
     */
    public TpResult notification(TpNotification notification) {
        TpResult result = masterRef.notification(notification);
        log.print(Constants.INFO, "Notification passed to Master.");
        return result;
    }
//---------------------------------------------------------------------------//
   /**
     * Provides a performance notification based on the checking of thresholds.
     * @param notification A performance notification.
     * @return The result of the method call.
     */
public TpResult autonomousNotification(TpAutonomousEvent autonomousNotification){
    TpResult result = masterRef.autonomousNotification(autonomousNotification);
    log.print(Constants.INFO, "Notification about critical value passed to Master.");
    return result;
}
/* END: MonitorEvent Interface */
/*BEGIN: SelectedDestination Interface */
//---------------------------------------------------------------------------//
/**
     * Provides the most suitable location for migration
     * @param the location where the Worker will migrate.
     */
public void setConditionsReport(String selectedlocation){
    log("Worker is trying to move to " +selectedlocation);
    GrasshopperAddress agencyAddress = new
GrasshopperAddress((String)readIPAddressesOfAvailableLocation()[readIPAddressesOfAvai
lableLocation().length-1]);
    agencyProxy =
(IAgentSystem)ProxyGenerator.newInstance(IAgentSystem.class,agencyAddress.generateAg
entSystemId(),agencyAddress);
        try   {
        agencyProxy.removeAgent(locationAgentId);
```

```
        log("LocationAgent removed with success...");      }
        catch(Exception e)  {
        log("Removing LocationAgent failed..");
        e.printStackTrace();  }
    state = 1;
        try {
        move(new GrasshopperAddress(selectedlocation));
        log("I am tying to move to the :" +selectedlocation);
        }
        catch (Exception e){
        log("Migration failed.");
        log("Failed to move to the :" +selectedlocation);
        e.printStackTrace();
        }
}
/*END : SelectedDestination Interface*/
//-------------------------------------------------------------------------//
    /**
     * Provides the appropriate monitor index.
     * @param The performance parameter name of a monitor.
     * @return The monitor index.
     */
    private int lookupMonitorIndex(String parameterName) {
        int i;
        for (i = 0; i < monitorList.length; i++) {
            if ((monitorList[i].getParameterName()).compareTo(parameterName) == 0) {
                break;
            }
        }
        return i;
    }
//-------------------------------------------------------------------------//
   /**
    * Provides the a reference to Diagnostics agent.
    * @return  A reference to the IExViewDiagnos interface
    */
public IExViewDiagnos getReferenceToDiagnosticsAgent(){
    AgentInfo[] DiagnosticsInfo;
    RegionProxy = getRegion();
    IExViewDiagnos DiagnosticsProxy = null;
    SearchFilter filter = new SearchFilter(SearchFilter.NAME +"=DiagnosticsAgent");
    DiagnosticsInfo = RegionProxy.listAgents(null, filter);
        if (DiagnosticsInfo.length != 0) {
            DiagnosticsProxy = (IExViewDiagnos)
ProxyGenerator.newInstance(IExViewDiagnos.class,
DiagnosticsInfo[0].getIdentifier().toString(), ProxyGenerator.SYNC);
        } else {            log("Cannot locate Diagnostics Agent.");        }
        if (DiagnosticsProxy == null) {  log("No Diagnostics Agent.");
        }
        return DiagnosticsProxy;
}
```

```
//---------------------------------------------------------------------------//
   /**
    * Provides the a reference to Repository
    * @return A reference to the IExViewRepos interface
    */
public IExViewRepos getReferenceToRepositoryAgent(){
    AgentInfo[] RepositoryInfo;
    RegionProxy = getRegion();
    IExViewRepos RepositoryProxy = null;
    SearchFilter filter = new SearchFilter(SearchFilter.NAME +"=RepositoryAgent");
    RepositoryInfo = RegionProxy.listAgents(null, filter);
        if (RepositoryInfo.length != 0) {
            RepositoryProxy = (IExViewRepos)
ProxyGenerator.newInstance(IExViewRepos.class,
RepositoryInfo[0].getIdentifier().toString(), ProxyGenerator.SYNC);
        } else {            log("Cannot locate Repository Agent.");      }
        if (RepositoryProxy == null) {  log("No Repository Agent.");        }
        return RepositoryProxy;
}
//---------------------------------------------------------------------------//
 public void ReportStatus(TpStatus Message){
   IExStatusReportView  WorkerExtentionAgentProxy;
      try{
        masterRef.ReportStatus(Message);
       log.print(Constants.INFO, "Report passed to Master.");
        }
      catch (Exception e) {
            log.print(Constants.ERROR, "Failed to pass report to the Master Agent");
            e.printStackTrace();
    }

    if (WorkerExtentionInfo!=null){
      try{
        WorkerExtentionAgentProxy = (IExStatusReportView)
ProxyGenerator.newInstance(IExStatusReportView.class,
WorkerExtentionInfo.getIdentifier().toString(), ProxyGenerator.SYNC);
        WorkerExtentionAgentProxy.ReportStatus(Message);
            }
      catch (Exception e) {
        log.print(Constants.ERROR, "Failed to pass report to the WorkerExtention  Agent");
        e.printStackTrace();
          }       }    }
//---------------------------------------------------------------------------//
  public void SetStatus(TpStatus Message){
   IExStatusReportView  WorkerExtentionAgentProxy;
      try{
        masterRef.SetStatus(Message);
       log.print(Constants.INFO, "Set Status passed to Master.");
        }
      catch (Exception e) {
            log.print(Constants.ERROR, "Failed to pass SetStatus  to the Master Agent");
```

```
            e.printStackTrace();
       }
       if (WorkerExtentionInfo!=null){
         try{
           WorkerExtentionAgentProxy =(IExStatusReportView)
ProxyGenerator.newInstance(IExStatusReportView.class,
WorkerExtentionInfo.getIdentifier().toString(), ProxyGenerator.SYNC);
           WorkerExtentionAgentProxy.ReportStatus(Message);          }
         catch (Exception e) {
           log.print(Constants.ERROR, "Failed to pass SetStatus to the WorkerExtention  Agent");
           e.printStackTrace();
             }       }    }
//-------------------------------------------------------------------------//
   public void failuresCounterUpdate(int FailureNumber){
     failures=FailureNumber;    }
//-------------------------------------------------------------------------//
public void MigrateToNode(int RequestID,Object TpMessage, String MessageType,
TpNodeToMigrate Message){
    this.RequestID= RequestID;
    this.TpMessage=TpMessage;
    this.MessageType=MessageType;
    Reattempt= true;
    String CurrentLocation = getInfo().getLocation().toString();
    TpAdaptationResult message = new TpAdaptationResult(new
TpAdaptationMessage("",counterMigrations,new TpErrorType("", getName())),true,"OK","",
new Object[2]);
        if
(CurrentLocation.equalsIgnoreCase(Message.NetworkLocation+"/InformationDesk")){
LocationToMigrate=Message.AlternativeNode+"/InformationDesk";       }
     if (CurrentLocation.equalsIgnoreCase(Message.AlternativeNode+"/InformationDesk")){
LocationToMigrate=Message.NetworkLocation+"/InformationDesk";     }
       if
(Message.NodeChosen==1&&!CurrentLocation.equalsIgnoreCase(Message.NetworkLocatio
n+"/InformationDesk")){
LocationToMigrate=Message.NetworkLocation+"/InformationDesk";     }
       if (Message.NodeChosen==0){
        LocationToMigrate=Message.AlternativeNode+"/InformationDesk";       }
           System.out.println("The NodeChosen      is        :" + Message.NodeChosen);
           System.out.println("The current location is       :" + CurrentLocation);
           System.out.println("The Message.NetworkLocation is :" +
Message.NetworkLocation);
           System.out.println("The Message.AlternativeNode is :" + Message.AlternativeNode);
           System.out.println("The LocationToMigrate       is :" + LocationToMigrate);
                   log("Trying to move to the address " + LocationToMigrate);
                   migrate =true;}
//-------------------------------------------------------------------------//
/* Removes the worker agent. */
/*********************************************************/
 private final class Finalizer implements Runnable, Serializable  {
       /* The thread used for the checking of the termination flag. */
       Thread thread;
```

```
      public synchronized void  start() {
         thread = new Thread(this, this.hashCode() + "Finalizer");
         thread.start();
      }
      public void run() {
         try {
            do {
               thread.sleep(100);     // Periodical check.
               if (terminate == true) {
                  remove();

                  break;
               }
            } while (terminate == false);

         } catch (Exception e) {
            e.printStackTrace();
         }       }    }
/*****************************************************************/
private final class Reaatempt implements Runnable, Serializable  {
 /* The thread used for initiating the reaatempt procedure. */
      Thread thread;
      public synchronized void  start() {
         thread = new Thread(this,"Reaatempt");
         thread.start();
      }
      public void run() {
         try {
           if (WorkerExtentionInfo!= null) {
            workerExtentionAgentRefeference  = (IWorkerToWorkerExtention)
ProxyGenerator.newInstance(IWorkerToWorkerExtention.class,
WorkerExtentionInfo.getIdentifier().toString(), ProxyGenerator.SYNC);
            } else {    log("Cannot locate Worker Extention Agent.");}                    try {
workerExtentionAgentRefeference.RestartReattempt(RequestID,TpMessage,MessageType);
}
   catch(Exception e){  log("Failed to restart reaatempt in WorkerExtention Agent.");
                    e.printStackTrace();    }
         } catch (Exception e) {
          log("Failed to start reaatemp procedure");
           e.printStackTrace();}          }
        Reattempt=false;
        RequestID=0;
        TpMessage= null;
        MessageType=null;
      }
   }
/*****************************************************************/
 private final class Migrator implements Runnable, Serializable  {
      /* The thread used for the checking of the termination flag. */
      Thread thread;
      public synchronized void  start() {
```

```
        thread = new Thread(this, this.hashCode() + "Migrator");
        thread.start();
    }
    public void run() {
        try {
            do {
                thread.sleep(10);     // Periodical check.
                if (migrate == true) {
                    boolean AgencyExists= pingAgency(new
GrasshopperAddress(LocationToMigrate));
                    System.out.println("The agency is  available :"+AgencyExists);
                    if (AgencyExists){ move (new GrasshopperAddress(LocationToMigrate));
}
                            else{
                           log("Abort migration because agency " +LocationToMigrate +" does not
exist  " );  }
                    break;
                }
            } while (migrate == false);
        } catch (Exception e) {
       log("Migration  failed to the  agency " +LocationToMigrate);
            e.printStackTrace();
        }      }    }
//*******************************************************
IWorkerToWorkerExtention workerExtentionAgentRefeference;
transient String LocationToMigrate;
boolean Reattempt= false;
int RequestID=0;
Object TpMessage=null;
String MessageType=null;
}
```

1. IInControllerAdaptationLogic

```
package uk.ac.surrey.callisto.pm.adaptation;
import uk.ac.surrey.callisto.pm.*;
import java.io.*;
/**
 * Supported by Adaptation Logic. It is used from the
 * Controller in order to pass the information needed from The
 * WorkerExtention Agent or Worker Agent to the Adaptation Logic.
 */
public interface IInControllerAdaptLogic {
   /**
    * It is invoked by the Controller when a Correspondent unavailability
    * failure occurs.
    * @param RequestID. An integer that defines the increasing number of the
    *                request.
    * @param Message. A data type that withholds information of the occurred
    *                failure (TpCollaboratorCorrespondentUnavailable)
    * @param ReferenceToDiagnostics. A reference to the Diagnostics agent
```

```
     *                          (interface IExViewDiagnos)
     * @param ReferenceToRepository.  A reference to the Repository agent
     *                          (interface IExViewRepos )
     */
  public void CorrespondentUnavailable(int RequestID,
                          TpCollaboratorCorrespondentUnavailable Message,
                          IExViewDiagnos ReferenceToDiagnostics,
                          IExViewRepos ReferenceToRepository );
    /**
     * It is invoked by the Controller when a Collaborator unavailability
     * failure occurs.
     * @param RequestID. An integer that defines the increasing number of the
     *               request.
     * @param Message. A data type that withholds information of the occurred
     *               failure (TpCollaboratorCorrespondentUnavailable)
     * @param ReferenceToDiagnostics. A reference to the Diagnostics agent
     *                          (interface IExViewDiagnos)
     * @param ReferenceToRepository.  A reference to the Repository agent
     *                          (interface IExViewRepos )
     */
public void CollaboratorUnavailable(int RequestID,
                          TpCollaboratorCorrespondentUnavailable Message,
                          IExViewDiagnos ReferenceToDiagnostics,
                          IExViewRepos ReferenceToRepository );
    /**
     * It is invoked by the Controller when a Resource unavailability
     * failure occurs.
     * @param RequestID. An integer that defines the increasing number of the
     *               request.
     * @param Message. A data type that withholds information of the occurred
     *               failure (TpResourceUnavailable)
     * @param ReferenceToDiagnostics. A reference to the Diagnostics agent
     *                          (interface IExViewDiagnos)
     * @param ReferenceToRepository.  A reference to the Repository agent
     *                          (interface IExViewRepos )
     */
public void ResourceUnavailable(int RequestID,
                        TpResourceUnavailable Message,
                        IExViewDiagnos ReferenceToDiagnostics,
                        IExViewRepos ReferenceToRepository );
    /**
     * It is invoked by the Controller when a Logic unavailability and Logic
     * update failures occur.
     *
     * @param RequestID. An integer that defines the increasing number of the
     *               request.
     * @param Message. A data type that withholds information of the occurred
     *               failure (TpLogicUnavailable)
     * @param ReferenceToDiagnostics. A reference to the Diagnostics agent
     *                          (interface IExViewDiagnos)
     * @param ReferenceToRepository.  A reference to the Repository agent
```

```
    *                       (interface IExViewRepos )
    */
 public void LogicUnavailable(int RequestID,
                        TpLogicUnavailable Message,
                        IExViewDiagnos ReferenceToDiagnostics,
                        IExViewRepos ReferenceToRepository );
   /**
    * It is invoked by the Controller when Proximity failures occur.
    * @param RequestID. An integer that defines the increasing number of the
    *                request.
    * @param Message. A data type that withholds information of the occurred
    *                failure (TpProximityFailure)
    * @param ReferenceToDiagnostics. A reference to the Diagnostics agent
    *                       (interface IExViewDiagnos)
    * @param ReferenceToRepository.  A reference to the Repository agent
    *                       (interface IExViewRepos )
    */
 public void ProximityFailure(int RequestID,
                        TpProximityFailure Message,
                        IExViewDiagnos ReferenceToDiagnostics,
                        IExViewRepos ReferenceToRepository );
   /**
    * It is invoked by the Controller when operational resources
    * unavailability failure occurs.
    * @param RequestID. An integer that defines the increasing number of the
    *                request.
    * @param Message. A data type that withholds information of the occurred
    *                failure (TpOperationalResourcesUtilization)
    * @param ReferenceToDiagnostics. A reference to the Diagnostics agent
    *                       (interface IExViewDiagnos)
    * @param ReferenceToRepository.  A reference to the Repository agent
    *                       (interface IExViewRepos )
    */
 public void OperationalResourcesUnavailability(int RequestID,
                                    TpOperationalResourcesUtilization Message,
                                    IExViewDiagnos ReferenceToDiagnostics,
                                    IExViewRepos ReferenceToRepository );
   /**
    * It is invoked by the Controller when the Worker Agent is about to
    * migrate to another network node. This method terminates all the instances of
    * adaptation logic classes that are active due to reaatempt method invocation.
    */
public void terminate();
  }
```

2. IInAdaptLogicView

```
package uk.ac.surrey.callisto.pm.adaptation;
import uk.ac.surrey.callisto.pm.*;
import java.io.*;
/**
```

```
 * Supported by View module in order to communicate
 * with the Adaptation Logic module
 */
public interface IInAdaptLogicView{
   /**
    * It is invoked by the Adaptation logic module when a Collaborator
    * unavailability failure occurs.
    * @param RequestID. An integer that defines the increasing number of the
    *                request.
    * @param Message. A data type that withholds information of the occurred
    *               failure (TpCollaboratorCorrespondentUnavailable)
    * @param ReferenceToDiagnostics. A reference to the Diagnostics agent
    *                          (interface IExViewDiagnos)
    * @param ReferenceToRepository.  A reference to the Repository agent
    *                          (interface IExViewRepos)
    * @return TpAdaptationResult. Contains information of the Adaptation
    *                         procedure.
    */
   public TpAdaptationResult RequestReportFailureCorrespondent(int RequestID,
                                          TpCollaboratorCorrespondentUnavailable Message,
                                          IExViewDiagnos ReferenceToDiagnostics,
                                          IExViewRepos ReferenceToRepository );
   /**
    * It is invoked by the Adaptation logic module when a proximity
    * failure occurs.
    * @param RequestID. An integer that defines the increasing number of the
    *                request.
    * @param Message. A data type that withholds information of the occurred
    *               failure (TpProximityFailure)
    * @param ReferenceToDiagnostics. A reference to the Diagnostics agent
    *                          (interface IExViewDiagnos)
    * @param ReferenceToRepository.  A reference to the Repository agent
    *                          (interface IExViewRepos)
    * @return TpAdaptationResult. Contains information of the Adaptation
    *                         procedure.
    */

   public TpAdaptationResult RequestReportFailureProximity(int RequestID,
                                        TpProximityFailure Message,
                                        IExViewDiagnos ReferenceToDiagnostics,
                                        IExViewRepos ReferenceToRepository );
   /**
    * It is invoked by the Adaptation logic module when critical resource
    * utilization occurs.
    * @param RequestID. An integer that defines the increasing number of the
    *                request.
    * @param Message. A data type that withholds information of the occurred
    *               failure (TpProximityFailure)
    * @param ReferenceToDiagnostics. A reference to the Diagnostics agent
    *                          (interface IExViewDiagnos)
    * @param ReferenceToRepository.  A reference to the Repository agent
```

```
     *                          (interface IExViewRepos)
     * @return TpAdaptationResult. Contains information of the Adaptation
     *                      procedure.
     */
public TpAdaptationResult RequestReportCriticalResourceUtilization
(int RequestID,  TpOperationalResourcesUtilization Message,
IExViewDiagnos ReferenceToDiagnostics, IExViewRepos ReferenceToRepository );
      /**
     * It is invoked by the Adaptation logic module when a resource unavailability
     * failure occurs.
     *
     * @param RequestID. An integer that defines the increasing number of the
     *               request.
     * @param Message. A data type that withholds information of the occurred
     *               failure (TpCollaboratorCorrespondentUnavailable)
     * @param ReferenceToDiagnostics. A reference to the Diagnostics agent
     *                          (interface IExViewDiagnos)
     * @param ReferenceToRepository.  A reference to the Repository agent
     *                          (interface IExViewRepos)
     * @return TpAdaptationResult. Contains information of the Adaptation
     *                      procedure.
     */
    public TpAdaptationResult ResourceUnavailable(int RequestID,
                                    TpResourceUnavailable Message,
                                    IExViewDiagnos ReferenceToDiagnostics,
                                    IExViewRepos ReferenceToRepository );
    /**
     * Orders the Worker agent to migrate to another netwrk node. It also
     * provides the required information for the failure that had as result Worker
     * agent's migration. After the migration the Worker uses this information in
     * order to resolve this failure.
     *
     * @param RequestID. An integer that defines the increasing number of the
     *               request.
     * @param TpMessage. A data type that is passed as object
     *                instance. Controller uses the string DataType
     *                in order to cast this object to the appropriate
     *                data type.
     * @param MessageType. A string that defines the data type of the
     *                  FailureParameters object.
     * @param Message. A TpNodeToMigrate data type.
     * @param ReferenceToDiagnostics. A reference to the Diagnostics agent
     *                          (interface IExViewDiagnos)
     * @param ReferenceToRepository.  A reference to the Repository agent
     *                          (interface IExViewRepos )
     */
    public  void MigrateToNode(int RequestID,
                        Object TpMessage,
                        String MessageType,
                        TpNodeToMigrate Message,
                        IExViewDiagnos ReferenceToDiagnostics,
```

```
                        IExViewRepos ReferenceToRepository );
    /**
     * Orders the  creation of an entity.
     * @param RequestID. An integer that defines the increasing number of the
     *                request.
     * @param Entity. A TpCreateEntity data type that specifies the
     *              entity to be created by the Repository agent.

     * @param ReferenceToRepository.  A reference to the Repository agent
     *                          (interface IExViewRepos )
     * @return TpAdaptationResult. Contains information of the Adaptation
     *                         procedure.
     */
    public TpAdaptationResult CreateEntity(int RequestID,
                                TpCreateEntity Entity,
                                IExViewRepos ReferenceToRepository );

    public TpAdaptationResult AdaptationCommand(int RequestID,Object Message);
    /**
     * Reports an adaptation result to View module in order to be forwarded
     * from the latter to the Worker Agent
     * @param Message. Is of TpStatus data type.
     */
    public void ReportStatus(TpStatus Message,IExViewDiagnos ReferenceToDiagnostics);
    /**
     * Reports an adaptation result to to View module in order to be forwarded
     * from the latter to the Worker Agent and
     * sets the current status to the WorkerExtentionAgent
     * @param Message. Is of TpStatus data type.
     * @param ReferenceToDiagnostics. A reference to the Diagnostics agent
     *                          (interface IExViewDiagnos)
     */
    public void SetStatus(TpStatus Message,IExViewDiagnos ReferenceToDiagnostics);
}
```

3. IExControllerNetMang

```
package uk.ac.surrey.callisto.pm.adaptation;
import uk.ac.surrey.callisto.pm.*;
import java.io.*;
/**
 *  It is supported by the Controller. It is used from the
 *  WorkerExtentionAgent in order to pass information relevant to the occurred
 *  failures.
 */
public interface IExControllerNetMang {
    /**
     * It is invoked by the WorkerExtentionAgent.
     * @return Object the
     */
public Object getStatus();
```

```
   /**
    * It is invoked by the WorkerExtentionAgent in order to set the status
    * of the system to the Controller.
    * @param Status. Is a string that states the current status of the
    * Worker (failure that occurred in the Network Performance System)
    */
public void setStatus(String Status);
   /**
    * It is invoked by the WorkerExtentionAgent in order to pass to the
    * Controler the data of the occurred failure.
    * @param FailureParameters. A data type that is passed as object
    *                        instance. Controller uses the string DataType
    *                        in order to cast this object to the appropriate
    *                        data type.
    * @param DataType. A string that defines the data type of the
    *                FailureParameters object.
    * @param ReferenceToDiagnostics. A reference to the Diagnostics agent
    *                           (interface IExViewDiagnos)
    * @param ReferenceToRepository.  A reference to the Repository agent
    *                           (interface IExViewRepos )
    */

public void setFailureParameters(Object FailureParameters,
                        String DataType,
                        IExViewDiagnos ReferenceToDiagnostics,
                        IExViewRepos ReferenceToRepository );
   /**
    * It is invoked by the WorkerAgent when the Worker Agent is about to
    * migrate to another network node. This method terminates all the instances of
    * adaptation logic classes that are active due to reattempt method invocation.
    */
public void TerminateAdaptationLogicModules();
  }
```

4. IExViewRepos

```
package uk.ac.surrey.callisto.pm.adaptation;
import uk.ac.surrey.callisto.pm.*;
import java.io.*;
/**
 * It is implemented by the Repository agent in order to communicate
 * with the View module.
 *
 */
 public interface IExViewRepos {
   /**
    * Orders the  creation of an entity.
    * @param RequestID. An integer that defines the increasing number of the
    *                request.
    * @param Entity. A TpCreateEntity data type that specifies the
    *              entity to be created by the Repository agent.
```

```
     * @param ReferenceToRepository.  A reference to the Repository agent
     *                          (interface IExViewRepos )
     * @return boolean.
     */
    public boolean CreateEntity(int RequestID,TpCreateEntity Entity);
    /**
     * Orders the  creation of a Worker Agent     */
    public void CreateWorker();
    /**
     * Provides the information needed for the creation of a Worker Agent.
     * @param Arguments. A TpCreationArgumements data type.
     */
    public void ArgumentsForWorkerCreation(TpCreationArgumements Arguments);
}
```

5. IWorkerToWorkerExtention

```
package uk.ac.surrey.callisto.pm.adaptation;
import uk.ac.surrey.callisto.pm.*;
/** It is supported from the WorkerExtentionAgent */
public interface IWorkerToWorkerExtention {
    /**
     * Instructs the WorkerExtentionAgent to start a reaatempt process
     * for a failure that had occurred before the Worker agent migration.
     * It is invoked by the Worker after its migration to an alternative node.
     * @param RequestID. An integer that defines the increasing number of the
     *                request.
     * @param TpMessage. A data type that is passed as object
     *                  instance. Controller uses the string DataType
     *                  in order to cast this object to the appropriate
     *                  data type.
     * @param MessageType. A string that defines the data type of the
     *                   FailureParameters object.
    */
public void RestartReattempt(int RequestID,
                        Object TpMessage,
                        String MessageType);
}
```

6. IExViewDiagnos

```
package uk.ac.surrey.callisto.pm.adaptation;
import uk.ac.surrey.callisto.pm.*;
import de.ikv.grasshopper.type.AgentInfo;
import java.io.*;
/**
 * It is implemented by the Diagnostics agent in order to communicate
 * with the View module.
 */
public interface IExViewDiagnos {
    /**
```

```
     * It is invoked by the View module in order for the Diagnostics agent
     * to identify if a failure of correspondent is occurred.
     * @param RequestID. An integer that defines the increasing number of the
     *                request.
     * @param Message. A data type that withholds information of the occurred
     *               failure (TpCollaboratorCorrespondentUnavailable)
     * @return TpAdaptationResult. Contains information of the Adaptation
     *                        procedure.
     */
public TpAdaptationResult IdentifyFailureCorrespondent(int
RequestID,TpCollaboratorCorrespondentUnavailable Message);
    /**
     * It is invoked by the View module in order for the Diagnostics agent
     * to identify if a proximity failure is occurred.
     * @param RequestID. An integer that defines the increasing number of the
     *                request.
     * @param Message. A data type that withholds information of the occurred
     *               failure (TpProximityFailure)
     * @return TpAdaptationResult. Contains information of the Adaptation
     *                        procedure.
     */
public TpAdaptationResult IdentifyProximityFailure(int RequestID,TpProximityFailure
Message);
    /**
     * It is invoked by the View module in order for the Diagnostics agent
     * to identify if critical resource utilization is occurred.
     * @param RequestID. An integer that defines the increasing number of the
     *                request.
     * @param Message. A data type that withholds information of the occurred
     *               failure (TpOperationalResourcesUtilization)
     * @return TpAdaptationResult. Contains information of the Adaptation
     *                        procedure.
     */
    public TpAdaptationResult IdentifyCriticalResourceUtilization (int RequestID,
                                                TpOperationalResourcesUtilization Message);
    /**
     * It is invoked by the View module in order for the Diagnostics agent
     * to identify if resource unavailability is occurred.
     * @param RequestID. An integer that defines the increasing number of the
     *                request.
     * @param Message. A data type that withholds information of the occurred
     *               failure (TpResourceUnavailable)
     * @return TpAdaptationResult. Contains information of the Adaptation
     *                        procedure.
     */
    public TpAdaptationResult IdentifyResourceUnavailable(int RequestID,
                                            TpResourceUnavailable Message);
    /**
     * It is invoked by the View module in order to get a reference to the
     * Worker agent
     * @return AgentInfo []. An array that contains information about the Worker agent
```

```
    */
   public AgentInfo[] getReferenceToWorker();
}
```

7. IExViewCarrier

```
package uk.ac.surrey.callisto.pm.adaptation;
import uk.ac.surrey.callisto.pm.*;
import java.io.*;
/**
 * It is implemented from the Worker agent in order to receive migration
 * requests from the Adaptation logic.
 */
public interface IExViewCarrier {
   /**
    * Orders the Worker agent to migrate to another netwrk node. It also
    * provides the required information for the failure that had as result Worker
    * agent's migration. After the migration the Worker uses this information in
    * order to resolve this failure.
    * @param RequestID. An integer that defines the increasing number of the
    *                request.
    * @param TpMessage. A data type that is passed as object
    *                instance. Controller uses the string DataType
    *                in order to cast this object to the appropriate
    *                data type.
    * @param MessageType. A string that defines the data type of the
    *                  FailureParameters object.
    * @param Message. A TpNodeToMigrate data type.
    */
   public void MigrateToNode(int RequestID,
                        Object TpMessage,
                        String MessageType,
                        TpNodeToMigrate Message);
}
```

8. IExTargetFailure

```
package uk.ac.surrey.callisto.pm.adaptation;
import uk.ac.surrey.callisto.pm.*;
import de.ikv.grasshopper.type.AgentInfo;
import java.io.*;
/** Supported by WorkerExtentionAgent in order to be notified when a Target
 agent has failed.  */

public interface IExTargetFailure {
   /**
    * Notifes the WorkerExtentionAgent that a Target agent has failed
    * @param TargetInfo. The AgentInfo of the Target Agent
    */
   public void TargetIsGoingDown(AgentInfo TargetInfo);}
```

9. IExStatusReportView

```
package uk.ac.surrey.callisto.pm.adaptation;
import uk.ac.surrey.callisto.pm.*;
import java.io.*;
/** It is implemented by the Worker agent in order to receive
 * reports concerning the adaptation process results
 */
public interface IExStatusReportView{
    /**
     * Reports an adaptation result to the Worker Agent
     * @param Message. Is of TpStatus data type.
     */
    public void ReportStatus(TpStatus Message);
    /**
     * Reports an adaptation result to the Worker Agent and
     * sets the current status to the WorkerExtentionAgent
     * @param Message. Is of TpStatus data type.
     */
    public void SetStatus(TpStatus Message);
}
```

10. IExPrintStatusMessages

```
package uk.ac.surrey.callisto.pm.adaptation;
import uk.ac.surrey.callisto.pm.*;
import java.io.*;
/**
 * Supported by WorkerExtention Agent. The entities that
 * wish to display messages to the MainFailures GUI
 * use this interface.
 */
 public interface IExPrintStatusMessages {
    /**
     * Displays messages to the MainFailures GUI
     * @param MessageToDisplay. A string that will be
     *                      displayed to the MainFailures GUI
     */
    public void DisplayMessageOnMainFailuresGUI (String MessageToDisplay);
}
```

11. IExLogic

```
package uk.ac.surrey.callisto.pm.adaptation;
import uk.ac.surrey.callisto.pm.*;
import java.io.*;
import java.util.Date;
/** Supported by Logic Agent. The interesting
 * entities can communicate with the Logic agent
```

```
 * by using this interface.*/
public interface IExLogic {

    /** This method return to the calling entity
     * the creation date of the Logic agent
     * @return Date. The date  of logic creation.
     */
  public Date getCreationTime();
}
```

12. IExDiagnosTarget

```
package uk.ac.surrey.callisto.pm.adaptation;
import uk.ac.surrey.callisto.pm.*;
import java.io.*;
/**
 * Supported by Target Agent in order for it to be able
 * to receive and reply to reports regarding management tasks
 */
public interface IExDiagnosTarget {
    /**
     * It is invoked by the Diagnostics agent in order for the Target agent
     * to identify if a failure of correspondent is occurred.
     * @param RequestID. An integer that defines the increasing number of the
     *                request.
     * @param Message. A data type that withholds information of the occurred
     *               failure (TpCollaboratorCorrespondentUnavailable)
     * @return TpAdaptationResult. Contains information of the Adaptation
     *                        procedure.
     */
    public TpAdaptationResult ReportFailureCorrespondent(int RequestID,
                                        TpCollaboratorCorrespondentUnavailable Message);
    /**
     * It is invoked by the Diagnostics agent in order for the Target agent
     * to identify if a proximity failure is occurred.
     * @param RequestID. An integer that defines the increasing number of the
     *                request.
     * @param Message. A data type that withholds information of the occurred
     *               failure (TpProximityFailure)
     *
     * @return TpAdaptationResult. Contains information of the Adaptation
     *                        procedure.
     */
public TpAdaptationResult ReportProximityFailure(int RequestID,TpProximityFailure
Message);
      /**
     * It is invoked by the Diagnostics agent in order for the Target agent
     * to identify if critical resource utilization is occurred.
     * @param RequestID. An integer that defines the increasing number of the
     *                request.
```

```
     * @param Message. A data type that withholds information of the occurred
     *                failure (TpOperationalResourcesUtilization)
     * @return TpAdaptationResult. Contains information of the Adaptation
     *                          procedure.
     */
    public TpAdaptationResult ReportCriticalResourceUtilization(int RequestID,
                                                TpOperationalResourcesUtilization Message);
    /** It is invoked by the Diagnostics agent in order for the Target agent
     * to identify if resource unavailability is occurred.
     * @param RequestID. An integer that defines the increasing number of the
     *                  request.
     * @param Message. A data type that withholds information of the occurred
     *                failure (TpResourceUnavailable)
     * @return TpAdaptationResult. Contains information of the Adaptation
     *                          procedure.
     */
    public TpAdaptationResult ReportResourceUnavailable(int RequestID,
                                            TpResourceUnavailable Message);
    /** The invocation of this method has as result the failure of the
    * Target agent  */
    public void SimulatedTargetFailure();
}
```

13. TpAdaptationMessage

```
package uk.ac.surrey.callisto.pm.adaptation;
import uk.ac.surrey.callisto.pm.*;
import java.io.*;
import java.lang.String;
/**
 * Defines the Adaptation Message. All other data types extending this class.
*/
public class TpAdaptationMessage implements Serializable {
    /**
    *Identifies the failure that is forwarded from Worker Extention Agent to
    *Intelligent Adaptation Module for further actions to be taken.
    */
    public  String  ID;
    /** An increasing number that identifies the Message. In this respect,
    * several messages can be pending in the system and to be dealt with seperately
    * within the adaptation system.
    */
    public  int MessageID;
    /** Identifies the type of the Error occured and the reporting entity.
    */
    public  TpErrorType Error;
    /** Force creation of the data type with required parameters only.    */
    public TpAdaptationMessage() { ID = null; MessageID = 1; Error = null; }
    /**
     * Creates TpAdaptationMessage data type.
```

```
    * @param ID.Identifies the type of the error.
    * @param MessageID The increasing number that identifies this Message -Failure
    * @param Error The error identificataion.     */
public TpAdaptationMessage(String  ID,
                           int  MessageID,
                           TpErrorType Error) {
      this.ID = ID;
      this.MessageID = MessageID;
      this.Error = Error;
   }
/*-------------------------------------------------------------------------*/
    /** This method is used to print the values of the data type.    */
public void ToString(){
   System.out.println(this.ID);
   System.out.println(this.MessageID);
   Error.ToString();
   }
/*-------------------------------------------------------------------------*/
}
```

14. TpAdaptationResult

```
package uk.ac.surrey.callisto.pm.adaptation;
import uk.ac.surrey.callisto.pm.*;
import java.io.*;
/** Defines the TpAdaptationResult data type. */
public class TpAdaptationResult extends TpAdaptationMessage implements Serializable {
   /** Specifies if the result of the adaptation procedure was successful or not. Has true or
false values.    */
  public final boolean Result;
   /**  Specifies the status after the adaptation procedure.    */
  public final String Status;
   /**  Specifies the adaptation method that has been followed.  */
   public final String MethodFollowed;
 /** Specifies any additional data that has to be considered from Adaptation logic.*/
   public final Object [] AdaptationData;
   /** Creates a TpAdaptationResult data type.
  * @param Message. A TpAdaptationMessage data type
    * @param Result. Specifies if the result of the adaptation procedure was successful or not.
Has true or false values.
    * @param Status. Specifies the status after the adaptation procedure.
    * @param MethodFollowed.Specifies the adaptation method that has been followed.
    * @param AdaptationData.Specifies any additional data that has to be considered from
Adaptation logic.      */
public TpAdaptationResult(
                  TpAdaptationMessage Message,
                  boolean Result,
                  String Status,
                  String MethodFollowed,
                  Object [] AdaptationData)
```

```
   {
    super.ID=Message.ID;
    super.MessageID=Message.MessageID;
    super.Error= Message.Error;
    this.Result = Result;
    this.Status = Status;
    this.MethodFollowed=MethodFollowed;
    this.AdaptationData=AdaptationData;
   }
/*---------------------------------------------------------------------*/
     /** This method is used to print the values of the data type.    */
 public void ToString(){
    super.ToString();
    System.out.println(this.Result);
    System.out.println(this.Status);
    System.out.println(this.MethodFollowed);
    System.out.println(this.AdaptationData);
  }
/*---------------------------------------------------------------------*/
}
```

15. TpStatus

```
package uk.ac.surrey.callisto.pm.adaptation;
import uk.ac.surrey.callisto.pm.*;
import java.io.*;
/** Defines the Status data type.*/
public class TpStatus extends TpAdaptationMessage implements Serializable {
   /** States the current state of the system after the adaptation procedure    */
      public final String PresentState;
   /** Specifies if a reattempt will be followed. Has true or false values  */
      public final boolean ReAttempt;
   /** Specifies in case of reattempt the time that will occur (in milliseconds) */
      public final double ReAttemptPeriod;
    /**
     * Creates a TpStatus data type.
     * @param Message.Is of TpAdaptationMessage data type.
     * @param PresentState. States the current state of the system after the
     * adaptation procedure
     * @param ReAttempt. Specifies if the process will be reattempted.
     * @param ReAttemptPeriod. Specifies in case of reattempt,
     * the time period that will occur(in milliseconds)
     */
public TpStatus(
              TpAdaptationMessage Message,
              String PresentState,
              boolean ReAttempt,
              double ReAttemptPeriod)
   {
       super.ID=Message.ID;
       super.MessageID=Message.MessageID;
```

```
        super.Error= Message.Error;
        this.PresentState = PresentState;
        this.ReAttempt = ReAttempt;
        this.ReAttemptPeriod = ReAttemptPeriod;
    }
/*-------------------------------------------------------------------------*/
     /** This method is used to print the values of the data type.    */
public void ToString(){
        super.ToString();
        System.out.println(this.PresentState);
        System.out.println(this.ReAttempt);
        System.out.println(this.ReAttemptPeriod);
  }
/*-------------------------------------------------------------------------*/
}
```

16. TpResourceUnavailable

```
package uk.ac.surrey.callisto.pm.adaptation;
import uk.ac.surrey.callisto.pm.*;
import java.io.*;
/** Defines the TpResourceUnavailable data type */
public class TpResourceUnavailable extends TpAdaptationMessage implements Serializable
{
   /**  Specifies the unavailable resource   */
      public String ResourceName;
     /**
      * Creates a TpResourceUnavailable data type
      * @param Message.Is of TpAdaptationMessage data type
      * @param ResourceName. Specifies the resource that is not available
      */
public TpResourceUnavailable( TpAdaptationMessage Message,
                              String ResourceName)
{
        super.ID=Message.ID;
        super.MessageID=Message.MessageID;
        super.Error= Message.Error;
        this.ResourceName=ResourceName;
    }
/*-------------------------------------------------------------------------*/
     /** This method is used to print the values of the data type.    */
public void ToString(){
        super.ToString();
        System.out.println(this.ResourceName);
  }
/*-------------------------------------------------------------------------*/
}
```

17. TpProximityFailure

```
package uk.ac.surrey.callisto.pm.adaptation;
import uk.ac.surrey.callisto.pm.*;
import java.io.*;
/** Defines the TpProximityFailure data type */
public class TpProximityFailure extends TpAdaptationMessage implements Serializable {
   /** Specifies the network location where the failured entity is located.   */
   public final String NetworkLocation;
   /** Specifies an alternative node that can be used instead of the failured node. */
   public final String AlternativeNode;
  /**  Specifies the name of the desired Grasshopper agency    */
   public final String AgencyName;
   /**  Specifies the reattempt period.   */
   public final long ReattemptPeriod;
   /** Specifies if migration of the Worker has been already performed.  */
   public boolean MigrationPerformed;
   /** Creates a TpProximityFailure data type
    * @param Message.Is of TpAdaptationMessage data type
    * @param NetworkLocation.Specifies the network location
    * where the failured entity is located.
    * @param AlternativeNode.Specifies an alternative node that can
    * be used instead
    * of the failured node.
    * @param ReattemptPeriod. Specifies the reattempt period (in milliseconds).
    * @param MigrationPerformed. Specifies if migration of the Worker has
    * been already performed.If it is true, the Worker agent reaatempts to
    * resolve the failure. If it is false Worker agent migrates to
    * the Alternative node.
    */
public TpProximityFailure(
                    TpAdaptationMessage Message,
                    String NetworkLocation,
                    String AlternativeNode,
                    String AgencyName,
                    long ReattemptPeriod,
                    boolean MigrationPerformed){
      super.ID=Message.ID;
      super.MessageID=Message.MessageID;
      super.Error= Message.Error;
      this.NetworkLocation = NetworkLocation;
      this.AlternativeNode= AlternativeNode;
      this.AgencyName= AgencyName;
      this.ReattemptPeriod= ReattemptPeriod;
      this.MigrationPerformed=MigrationPerformed;    }
/*-------------------------------------------------------------------------*/
    /** This method is used to print the values of the data type.    */
public void ToString(){
      super.ToString();
      System.out.println(this.NetworkLocation);
```

```
        System.out.println(this.AlternativeNode);
        System.out.println(this.AgencyName);
        System.out.println(this.ReattemptPeriod);
        System.out.println(this.MigrationPerformed);
   }
/*-------------------------------------------------------------------------*/
}
```

18. TpOperationalResourcesUtilization

```
package uk.ac.surrey.callisto.pm.adaptation;
import uk.ac.surrey.callisto.pm.*;
import java.io.*;
/**Defines the Adaptation Message for the communication
 * in case of Operational Resources failure.
 */
public class TpOperationalResourcesUtilization extends TpAdaptationMessage implements
Serializable {
    /**  Specifies the network location where the critical resource
    * utilization is occurred.  */
      public  String NetworkLocation;
    /**  Specifies an alternative node that can be used instead of the
    * failured node.    */
     public final String AlternativeNode;
    /**  States if some resources have to be released. Has true or false value. */
      public  final boolean ReleaseResource;
    /** Creates a TpOperationalResourcesUtilization data type.
     * @param Message.A TpAdaptationMessage data type
     * @param NetworkLocation.Specifies the network location where the critical
     * resource utilization is occurred.
     * @param AlternativeNode.Specifies an alternative node that can be used
     * instead of the failured node
     * @param ReleaseResource. States if some resources have to be released.  */
public TpOperationalResourcesUtilization(TpAdaptationMessage Message,
                                String NetworkLocation,
                                String AlternativeNode,
                                boolean ReleaseResource){
        super.ID=Message.ID;
        super.MessageID=Message.MessageID;
        super.Error= Message.Error;
        this.NetworkLocation = NetworkLocation;
        this.AlternativeNode = AlternativeNode;
        this.ReleaseResource = ReleaseResource;}
/*-------------------------------------------------------------------------*/
     /** This method is used to print the values of the data type.    */
 public void ToString(){
        super.ToString();
        System.out.println(this.NetworkLocation);
        System.out.println(this.AlternativeNode);
        System.out.println(this.ReleaseResource);
  }/*-------------------------------------------------------------------------*/    }
```

19. TpNodeToMigrate

```
package uk.ac.surrey.callisto.pm.adaptation;
import uk.ac.surrey.callisto.pm.*;
import java.io.*;
import java.lang.String;
/**  Defines the TpNodeToMigrate data type.*/
public class TpNodeToMigrate implements Serializable {
   /** Specifies the network location where the Worker agent is to migrate to.   */
    public final String NetworkLocation;
    /** Specifies an alternative node that can be used instead of the failured node.   */
    public final String AlternativeNode;
    /** Specifies which of the nodes (NetworkLocation or Alternative) the
    * Worker agent will migrate to.Its value (0 or 1) is determined by the
    * adaptation result    */
    public final int NodeChosen;
      /** Creates a TpNodeToMigrate data type.
     * @param NetworkLocation.Specifies the network location where the Worker agent is to
migrate to
     * @param AlternativeNode.Specifies an alternative node that can be used
     * instead of the failured node.
     * @param NodeChosen.Specifies which of the nodes (NetworkLocation or Alternative)
the Worker agent will migrate to.
     */
public TpNodeToMigrate(String NetworkLocation,String AlternativeNode, int NodeChosen
){
        this.NetworkLocation = NetworkLocation;
        this.AlternativeNode=AlternativeNode;
        this.NodeChosen= NodeChosen;       }
/*-------------------------------------------------------------------------*/
     /** This method is used to print the values of the data type.    */
public void ToString(){
        System.out.println(this.NetworkLocation);
        System.out.println(this.AlternativeNode);
        System.out.println(this.NodeChosen);
  }/*-------------------------------------------------------------------------*/    }
```

20. TpLogicUnavailable

```
package uk.ac.surrey.callisto.pm.adaptation;
import uk.ac.surrey.callisto.pm.*;
import java.io.*;
/** Defines the message needed for the communication exchange in the case
 * of Logic is not available or Logic update is required.  */
public class TpLogicUnavailable extends TpAdaptationMessage implements Serializable {
    /** Specifies the network location where the logic is located. */
    public final String NetworkLocation;


    /*** Creates a TpLogicUnavailable data type.
```

```
    * @param Message.A TpAdaptationMessage data type
    * @param NetworkLocation.Specifies the network location where
    * the logic is located.
    */
public TpLogicUnavailable(
                TpAdaptationMessage Message,
                String NetworkLocation) {
       super.ID=Message.ID;
       super.MessageID=Message.MessageID;
       super.Error= Message.Error;
       this.NetworkLocation=NetworkLocation;  }
/*-------------------------------------------------------------------------*/
    /** This method is used to print the values of the data type.    */
public void ToString(){
       super.ToString();
       System.out.println(this.NetworkLocation);
  }/*-------------------------------------------------------------------------*/
}
```

21. TpErrorType

```
package uk.ac.surrey.callisto.pm.adaptation;
import uk.ac.surrey.callisto.pm.*;
import java.io.*;
import java.lang.String;
/** Defines the Error Type data type.*/
public class TpErrorType implements Serializable {
    /** Provides a short description of the occurred failure.  */
    public final String ErrorDescription;
   /**Specifies the entity reporting the failure.   */
    public final String ReportingEntity;
/*-------------------------------------------------------------------------*/
    /** Forces the creation of a default error type.  */
private TpErrorType() {
       ErrorDescription = null; ReportingEntity = null;
    }
/*-------------------------------------------------------------------------*/
   /**Creates a TpErrorType data type.
    *@param ErrorDescription.Provides a short description of the
    *occurred failure.
    *@param  ReportingEntity.Specifies the entity reporting the failure.
    */
public TpErrorType(String ErrorDescription, String ReportingEntity) {
       this.ErrorDescription = ErrorDescription;
       this.ReportingEntity = ReportingEntity;        }

/*-------------------------------------------------------------------------*/
    /** This method is used to print the values of the data type.    */
public void ToString(){

       System.out.println(this.ErrorDescription);
```

```
        System.out.println(this.ReportingEntity);
    }/*-------------------------------------------------------------------------*/
     /**Returns the description of the specified error type.    */
  public String ReturnErrorName(){
    return ErrorDescription;
    }/*-------------------------------------------------------------------------*/
}
```

22. TpCreateEntity

```
package uk.ac.surrey.callisto.pm.adaptation;
import uk.ac.surrey.callisto.pm.*;
import java.io.*;
import java.lang.String;
/* Defines TpCreateEntity data type. */
public class TpCreateEntity implements Serializable {
    /** Specifies the entity to be created by the repository agent.  */
    public final String Entity;
    /** Specifies the network location where this entity has to be sent after
    * the creation from the repository agent. */
    public final String NetworkLocation;
    /** Creates a TpCreateEntity data type.
     * @param Entity.Specifies the entity to be created by the repository agent.
     * @param NetworkLocation. Specifies the network location where this entity
     * has to be sent after creation from the repository agent.     */
public TpCreateEntity(String Entity,String NetworkLocation){
        this.Entity= Entity;
        this.NetworkLocation = NetworkLocation;
    }
/*-------------------------------------------------------------------------*/
/**This method is used to print the values of this data type.*/
public void ToString(){
        System.out.println(this.Entity);
        System.out.println(this.NetworkLocation);
  }/*-------------------------------------------------------------------------*/}
```

23. TpCollaboratorCorrespondentUnavailable

```
package uk.ac.surrey.callisto.pm.adaptation;
import uk.ac.surrey.callisto.pm.*;
import java.io.*;
import java.lang.String;
/** Defines the message needed for the communication exchange in case
 * of migration failures or Remote Communication faulure.*/
public class TpCollaboratorCorrespondentUnavailable extends TpAdaptationMessage
implements Serializable {
    /** Specifies the network location where the Collaborator or Correspondent failure
occurred.    */
    public final String NetworkLocation;
    /** Specifies the reattempt period that will be followed if the adaptation
```

```
   * procedure ends unsuccesfully(in milliseconds). */
   public final double ReattemptPeriod;
   /** * Creates a TpCollaboratorCorrespondentUnavailable data type. This data
    * type is used for migration failures and in Remote Communication faulure.
    * @param Message.A TpAdaptationMessage data type
    * @param NetworkLocation.Specifies the network location where the Collaborator
    * or Correspondent failure occurred.
    * @param ReattemptPeriod. Specifies the reattempt period that will be followed if
    * the adaptation procedure ends unsuccesfully(in milliseconds).
    */
public TpCollaboratorCorrespondentUnavailable(
                              TpAdaptationMessage Message,
                              String NetworkLocation,
                              int ReattemptPeriod){
      super.ID=Message.ID;
      super.MessageID=Message.MessageID;
      super.Error= Message.Error;
      this.NetworkLocation = NetworkLocation;
      this.ReattemptPeriod= ReattemptPeriod; }
/** This method is used to print the values of the data type.    */
public void ToString(){
      super.ToString();
      System.out.println(this.NetworkLocation);
      System.out.println(this.ReattemptPeriod);   }
/*-------------------------------------------------------------------------*/  }
```

24. WorkerExtentionAgent

```
package uk.ac.surrey.callisto.pm.adaptation;
import uk.ac.surrey.callisto.pm.adaptation.*;
import uk.ac.surrey.callisto.pm.*;
import uk.ac.surrey.callisto.pm.adaptation.FailuresGUI.*;
import uk.ac.surrey.callisto.pm.adaptation.UpdateProperties.*;
import de.ikv.grasshopper.agent.*;
import de.ikv.grasshopper.agency.*;
import de.ikv.grasshopper.type.*;
import de.ikv.grasshopper.communication.*;
import de.ikv.grasshopper.util.*;
import java.util.*;
import java.io.*;
import java.net.*;
/**
 * The WorkerExtentionAgent initiating and controlling a number of monitoring tasks. Also
initiates and simulates failures to the Netowork management system. */
public class WorkerExtentionAgent extends MobileAgent implements IExTargetFailure,
Serializable, IExPrintStatusMessages,IExStatusReportView, IWorkerToWorkerExtention {
   String name;
   public  MainFailuresGUI    GUIReference= null;
   String [] Parameters = new String [3];
   transient IRegion RegionProxy ;
```

```
    private int counterMigrations=0;
    Log log= new Log();
    Vector ControllerCreated=new Vector();
//-------------------------------------------------------------------------//
    /** Empty object initialization. */
public WorkerExtentionAgent(){    }
//-------------------------------------------------------------------------//
  public void init(Object[] args) {
    CurrentArguments= args;
    CounterRequestID =((Integer)args[0]).intValue();
    System.out.println("The CounterRequestID is the following : "+CounterRequestID);}
//-------------------------------------------------------------------------//
    /** This method is used to read the IP addresses of participating servers
     * into the Monitoring System. This information is contained into the
     * file  ParticipatingServers.properties which is located on the path
     * C:/java/bin/lib/. The file ParticipatingServers.properties is updated
     * from the main GUI "Environment setup".
     * @return An array of Strings containing the IP of participating
     * servers.
     */
 public  String [] readIPAddressesOfAvailableLocation (){
    ReadWrite ReadFromFile = new ReadWrite();
    Vector  Servers = ReadFromFile.read("./lib/ParticipatingServers.properties");
    String [] ParticipatingServers= new String [Servers.size()];
                    try   {
           for( int index=0; index <(Servers.size());index++){
ParticipatingServers[index]="socket://"+(String)Servers.elementAt(index)+":7000/destination
";        }     }
        catch (Exception writeErr){writeErr.printStackTrace();      }
return ParticipatingServers; }
//-------------------------------------------------------------------------//
    /** Provide the agent name.
     * @return The WorkerExtentionAgent name  */
    public String getName() {
       name = "WorkerExtentionAgent";
       return name;   }
//-------------------------------------------------------------------------//
    /**
     * Provide an agent description.
     * @return The description of WorkerExtentionAgent      */
    public String getDescription() {
       return "WorkerExtentionAgent";    }
//-------------------------------------------------------------------------//
    /**
     * This method is called when the WorkerExtentionExtention
     * has migrated to a another network node.     */
public void afterMove (){
    if (GUIReference==null){
    runGUI();    }
    String [] ParticipatingServersRead = readIPAddressesOfAvailableLocation ();}
//-------------------------------------------------------------------------//
```

```
public void beforeRemove(){
   if (GUIReference!=null){
   GUIReference.setVisible(false);
   GUIReference.dispose();
   GUIReference= null;    }
   if (ControllerCreated.isEmpty()!=true){
   TerminateMonitors();    }
}
//-------------------------------------------------------------------------//
public void beforeMove(){
   GUIReference.setVisible(false);
   GUIReference.dispose();
   GUIReference= null;
   if (ControllerCreated.isEmpty()!=true){
   TerminateMonitors();    }}
//-------------------------------------------------------------------------//
public void action(){
runGUI();}
//-------------------------------------------------------------------------//
   /** Specify agent's behaviour.   */
public void live() {
   CounterRequestID =((Integer)CurrentArguments[0]).intValue();
   counterMigrations++;
   String [] ParticipatingServersRead = readIPAddressesOfAvailableLocation ();
   IExViewDiagnos ReferenceToDiagnostics = getReferenceToDiagnosticsAgent();
   IExViewRepos ReferenceToRepository = getReferenceToRepositoryAgent();
  if (GUIReference==null){
   runGUI();   }}
//-------------------------------------------------------------------------//
  public  void runGUI(){
      GUIReference=new MainFailuresGUI(this);;
            GUIReference.setVisible(true);  }
//-------------------------------------------------------------------------//
/**
* This method is called from the MainFailuresGUI class
*/
public void setParameters (String [] SelectedParameters){
        CounterRequestID++;
                    Parameters = SelectedParameters;
        if (Parameters[1].equalsIgnoreCase("Collaborator Unavailable")){
            boolean returnflag=TargetSimulatedFailure();
        } else {    initialization(Parameters);    }
}
/** Initiates the failure handling system.
*@param SelectedParameters[0]: "Simulation"
*@param SelectedParameters[1]: The failure
*@param SelectedParameters[2]: The arguments of the failure*/
public void initialization(String [] SelectedParameters){

  getReferenceToWorker().failuresCounterUpdate(CounterRequestID);
   try{
```

```
        SetCurrentStatus(SelectedParameters[1]);
        IExControllerNetMang ControllerObject = new  Controller(this);
        ControllerObject.setStatus(getCurrentStatus());
ControllerObject.setFailureParameters(FormDataType(),SelectedParameters[1],getReference
ToDiagnosticsAgent(),getReferenceToRepositoryAgent());
        ControllerCreated.addElement((IExControllerNetMang)ControllerObject);   }
  catch (Exception e){    e.printStackTrace();    }
   SelectedParameters= null;    }
//-------------------------------------------------------------------------//
public void ReportStatus(TpStatus Message){
   Calendar rightNow = Calendar.getInstance();
   DisplayMessageOnMainFailuresGUI("----------Reported Status-----------------");
   DisplayMessageOnMainFailuresGUI("Time of Report    :"+
rightNow.getTime().toString());
   DisplayMessageOnMainFailuresGUI("Failure number    :" + Message.MessageID);
   DisplayMessageOnMainFailuresGUI("Failure type      :" + Message.ID);
   DisplayMessageOnMainFailuresGUI("Present state is  :" + Message.PresentState);
   DisplayMessageOnMainFailuresGUI("Reattempt invoked :" + Message.ReAttempt);
   DisplayMessageOnMainFailuresGUI("Reaatempt period  :" + Message.ReAttemptPeriod);
   DisplayMessageOnMainFailuresGUI("-----------------------------------------");
if (Message.ReAttempt == true){
    SetCurrentStatus ("Reattempt| "+" Failure type : " + Message.MessageID+" Failure ID : "+
Message.ID );
 }}
//-------------------------------------------------------------------------//
public  void  SetStatus(TpStatus Message){
   Calendar rightNow = Calendar.getInstance();
   DisplayMessageOnMainFailuresGUI("---------Current status--------------------");
   DisplayMessageOnMainFailuresGUI("Time of Report
:"+rightNow.getTime().toString());
   DisplayMessageOnMainFailuresGUI("Failure number    :" + Message.MessageID);
   DisplayMessageOnMainFailuresGUI("Failure type      :" + Message.ID);
   DisplayMessageOnMainFailuresGUI("Present state is  :" + Message.PresentState);
   DisplayMessageOnMainFailuresGUI("Reattempt invoked :" + Message.ReAttempt);
   DisplayMessageOnMainFailuresGUI("Reaatempt period  :" + Message.ReAttemptPeriod);
   DisplayMessageOnMainFailuresGUI("-----------------------------------------");
   if (Message.ReAttempt == false){
   SetCurrentStatus ("Normal Conditions");
   }}
//-------------------------------------------------------------------------//
public static void SetCurrentStatus (String CurrentStatusUpdate){
 CurrentStatus = CurrentStatusUpdate;}
//-------------------------------------------------------------------------//
public static String getCurrentStatus(){
return CurrentStatus;}
//-------------------------------------------------------------------------//
public void RestartReattempt(int RequestID,Object TpMessage, String MessageType){
  Reaatempt= true;
  log("The MessageType is  :"+MessageType);
  if (MessageType.equalsIgnoreCase("RunTime
EnvironmentUnavailable")||MessageType.equalsIgnoreCase("Migration Denied")){
```

```
   TpProximityFailure ReAttempMessage=(TpProximityFailure)TpMessage;
   ReAttempMessage.MigrationPerformed=true;
   ReattemptRequest=ReAttempMessage.MessageID;
   log("Restaring reattempt method of the failure
:\n"+ReAttempMessage.ID+"\n"+ReAttempMessage.MessageID);
   String [] ReaatemptParameters = new String [3];
   ReaatemptParameters[0]="Simulation";
   ReaatemptParameters[1]= ReAttempMessage.ID;
   ReaatemptParameters[2]=
ReAttempMessage.NetworkLocation+","+ReAttempMessage.AlternativeNode+","+ReAttem
pMessage.ReattemptPeriod;
   //initialization(ReaatemptParameters);  }}
//-------------------------------------------------------------------------//
public  Object FormDataType(){
DataTypeTemlate = new Object();
String ResourceName =null;
String[] Arguments = new String [4];
   if (getCurrentStatus().equalsIgnoreCase("Correspondent Unavailable")){
      stopTheMaster();
      TpAdaptationMessage TpMessage = new
TpAdaptationMessage("Correspondent",CounterRequestID,new
TpErrorType("Correspondent","Worker"));
      TpCollaboratorCorrespondentUnavailable  DataTypeMessage = new
TpCollaboratorCorrespondentUnavailable(TpMessage,MasterInfo[0].getHome().toString(),1);
      DataTypeTemlate = (Object)DataTypeMessage;
  }
   if (getCurrentStatus().equalsIgnoreCase("Collaborator Unavailable")){
      TpAdaptationMessage TpMessage = new
TpAdaptationMessage("Collaborator",CounterRequestID,new
TpErrorType("Collaborator","Worker"));
      TpCollaboratorCorrespondentUnavailable  DataTypeMessage = new
TpCollaboratorCorrespondentUnavailable(TpMessage,FailedTarget.getHome().toString(),1);
      DataTypeTemlate = (Object)DataTypeMessage;
      FailedTarget= null;
    }
   if (getCurrentStatus().equalsIgnoreCase("Resource Unavailable")){
      TpAdaptationMessage TpMessage = new TpAdaptationMessage("Resource
Unavailable",CounterRequestID,new TpErrorType("Resource Unavailable","Worker"));
      ResourceName= Parameters[2];
         if (Parameters[2]==null){
         ResourceName= "alternative";
         DisplayMessageOnMainFailuresGUI("\nResource name has not been specified.
Default value will be used :" +ResourceName);          }
      TpResourceUnavailable DataTypeMessage = new
TpResourceUnavailable(TpMessage,ResourceName);
      DataTypeTemlate = (Object)DataTypeMessage;       }
   if (getCurrentStatus().equalsIgnoreCase("Logic Unavailable")){
      TpAdaptationMessage TpMessage = new TpAdaptationMessage("Logic
Unavailable",CounterRequestID,new TpErrorType("Logic Unavailable","Worker"));
      TpLogicUnavailable DataTypeMessage = new
TpLogicUnavailable(TpMessage,getInfo().getLocation().toString());
```

```
      DataTypeTemlate = (Object)DataTypeMessage;      }
     if (getCurrentStatus().equalsIgnoreCase("Migration Denied")){
      TpAdaptationMessage TpMessage;
         if (Parameters[2]==null){
         DisplayMessageOnMainFailuresGUI("\nArguments have not been specified.Default
values will be used.");            }
      Arguments = ConvertInputArguments(Parameters[2]);
      DisplayMessageOnMainFailuresGUI("-----INPUT ARGUMENTS----------------------");
      DisplayMessageOnMainFailuresGUI("Network Location     :  "+Arguments[0]);
      DisplayMessageOnMainFailuresGUI("Alternative Location :  "+Arguments[1]);
      DisplayMessageOnMainFailuresGUI("Agency to search     :  "+Arguments[2]);
      DisplayMessageOnMainFailuresGUI("Reattempt period     :  "+Arguments[3]);
      DisplayMessageOnMainFailuresGUI("-----------------------------------------");
      if (Reaatempt==true){
      TpMessage = new TpAdaptationMessage("Migration Denied",ReattemptRequest,new
TpErrorType("Migration Denied","Worker"));       }
      else {      TpMessage = new TpAdaptationMessage("Migration
Denied",CounterRequestID,new TpErrorType("Migration Denied","Worker"));      }
      TpProximityFailure DataTypeMessage = new
TpProximityFailure(TpMessage,Arguments[0],Arguments[1],Arguments[2],Long.parseLong(
Arguments[3]),Reaatempt);
      Reaatempt=false;
      DataTypeTemlate = (Object)DataTypeMessage;
      Arguments=null;       }

    if (getCurrentStatus().equalsIgnoreCase("RunTime Environment Unavailable")){
      TpAdaptationMessage TpMessage;
      if (Parameters[2]==null){
      DisplayMessageOnMainFailuresGUI("\nArguments have not been specified.Default
values will be used.");        }
      Arguments = ConvertInputArguments(Parameters[2]);
      DisplayMessageOnMainFailuresGUI("-----INPUT ARGUMENTS----------------------");
      DisplayMessageOnMainFailuresGUI("Network Location     :  "+Arguments[0]);
      DisplayMessageOnMainFailuresGUI("Alternative Location :  "+Arguments[1]);
      DisplayMessageOnMainFailuresGUI("Agency to search     :  "+Arguments[2]);
      DisplayMessageOnMainFailuresGUI("Reattempt period     :  "+Arguments[3]);
      DisplayMessageOnMainFailuresGUI("-----------------------------------------");
      if (Reaatempt==true){
      TpMessage = new TpAdaptationMessage("RunTime Environment
Unavailable",ReattemptRequest,new TpErrorType("RunTime Environment
Unavailable","Worker"));       }
      else {       TpMessage = new TpAdaptationMessage("RunTime Environment
Unavailable",CounterRequestID,new TpErrorType("RunTime Environment
Unavailable","Worker"));      }
      TpProximityFailure DataTypeMessage = new
TpProximityFailure(TpMessage,Arguments[0],Arguments[1],Arguments[2],Long.parseLong(
Arguments[3]),Reaatempt);
      Reaatempt= false;
      DataTypeTemlate = (Object)DataTypeMessage;
      Arguments=null;    }
  if (getCurrentStatus().equalsIgnoreCase("Network Node Inaccessible")){
```

```
      TpAdaptationMessage TpMessage;
      if (Parameters[2]==null){
      DisplayMessageOnMainFailuresGUI("\nArguments have not been specified.Default
values will be used.");       }
      Arguments = ConvertInputArguments(Parameters[2]);
      DisplayMessageOnMainFailuresGUI("-----INPUT ARGUMENTS----------------------");
      DisplayMessageOnMainFailuresGUI("Network Location     :  "+Arguments[0]);
      DisplayMessageOnMainFailuresGUI("Alternative Location :  "+Arguments[1]);
      DisplayMessageOnMainFailuresGUI("Agency to search     :  "+Arguments[2]);
      DisplayMessageOnMainFailuresGUI("Reattempt period     :  "+Arguments[3]);
      DisplayMessageOnMainFailuresGUI("----------------------------------------");
         if (Reaatempt==true){
            TpMessage = new TpAdaptationMessage("Network Node
Inaccessible",ReattemptRequest,new TpErrorType("Network Node Inaccessible","Worker"));
}
         else {            TpMessage = new TpAdaptationMessage("Network Node
Inaccessible",CounterRequestID,new TpErrorType("Network Node Inaccessible","Worker"));
}
      TpProximityFailure DataTypeMessage = new
TpProximityFailure(TpMessage,Arguments[0],Arguments[1],Arguments[2],Long.parseLong(
Arguments[3]),Reaatempt);
      Reaatempt= false;
      DataTypeTemlate = (Object)DataTypeMessage;
      Arguments=null;    }
  if (getCurrentStatus().equalsIgnoreCase("Remote Communication Failure")){
      TpAdaptationMessage TpMessage;
         if (Parameters[2]==null){
         DisplayMessageOnMainFailuresGUI("\nArguments have not been specified.Default
values will be used.");          }
      Arguments = ConvertInputArguments(Parameters[2]);
      DisplayMessageOnMainFailuresGUI("-----INPUT ARGUMENTS----------------------");
      DisplayMessageOnMainFailuresGUI("Network Location     :  "+Arguments[0]);
      DisplayMessageOnMainFailuresGUI("Alternative Location :  "+Arguments[1]);
      DisplayMessageOnMainFailuresGUI("Agency to search     :  "+Arguments[2]);
      DisplayMessageOnMainFailuresGUI("Reattempt period     :  "+Arguments[3]);
      DisplayMessageOnMainFailuresGUI("----------------------------------------");
         if (Reaatempt==true){
         TpMessage = new TpAdaptationMessage("Remote Communication
Failure",ReattemptRequest,new TpErrorType("Remote Communication Failure","Worker"));
}
         else {          TpMessage = new TpAdaptationMessage("Remote Communication
Failure",CounterRequestID,new TpErrorType("Remote Communication Failure","Worker"));
}
      TpProximityFailure DataTypeMessage = new
TpProximityFailure(TpMessage,Arguments[0],Arguments[1],Arguments[2],Long.parseLong(
Arguments[3]),Reaatempt);
      DataTypeTemlate = (Object)DataTypeMessage;
      Reaatempt= false;   }
  if (getCurrentStatus().equalsIgnoreCase("Logic Update Required")){
      TpAdaptationMessage TpMessage = new TpAdaptationMessage("Logic Update
Required",CounterRequestID,new TpErrorType("Logic Update Required","Worker"));
```

```
        TpLogicUnavailable  DataTypeMessage = new
TpLogicUnavailable(TpMessage,getInfo().getLocation().toString());
        DataTypeTemlate = (Object)DataTypeMessage;     }
  if (getCurrentStatus().equalsIgnoreCase("Critical Resource Utilization")){
    boolean releaseResouces = false;
      if (Parameters[2]==null){
        DisplayMessageOnMainFailuresGUI("\nArguments have not been specified.Default
values will be used.");       }
        Arguments = ConvertInputArgumentsOperationalResources(Parameters[2]);
        DisplayMessageOnMainFailuresGUI("-----INPUT ARGUMENTS----------------------");
        DisplayMessageOnMainFailuresGUI("Network Location     :  "+Arguments[0]);
        DisplayMessageOnMainFailuresGUI("Alternative Location :  "+Arguments[1]);
        DisplayMessageOnMainFailuresGUI("Release resources    :  "+Arguments[2]);
        DisplayMessageOnMainFailuresGUI("----------------------------------------");
        TpAdaptationMessage TpMessage = new TpAdaptationMessage("Critical Resource
Utilization",CounterRequestID,new TpErrorType("Critical Resource Utilization","Worker"));
        TpOperationalResourcesUtilization   DataTypeMessage = new
TpOperationalResourcesUtilization(TpMessage,Arguments[0],Arguments[1],Boolean.getBoo
lean(Arguments[2]));
        DataTypeTemlate = (Object)DataTypeMessage;      }
return DataTypeTemlate;}
//-------------------------------------------------------------------------//
public Object getStatus(){
    return DataTypeTemlate; }
//-------------------------------------------------------------------------//
public void MigrateToNode(int RequestID,TpCreateEntity Message){
  Message.ToString();}
//-------------------------------------------------------------------------//
public IExViewDiagnos getReferenceToDiagnosticsAgent(){
    AgentInfo[] DiagnosticsInfo;
    RegionProxy = getRegion();
    IExViewDiagnos DiagnosticsProxy = null;
    SearchFilter filter = new SearchFilter(SearchFilter.NAME +"=DiagnosticsAgent");
    DiagnosticsInfo = RegionProxy.listAgents(null, filter);
        if (DiagnosticsInfo.length != 0) {
            DiagnosticsProxy = (IExViewDiagnos)
ProxyGenerator.newInstance(IExViewDiagnos.class,
DiagnosticsInfo[0].getIdentifier().toString(), ProxyGenerator.SYNC);
        } else { log("Cannot locate Diagnostics Agent.");   }
        if (DiagnosticsProxy == null) {  log("No Diagnostics Agent.");    }
        return DiagnosticsProxy;}
//-------------------------------------------------------------------------//
public IExViewRepos getReferenceToRepositoryAgent(){
    AgentInfo[] RepositoryInfo;
    RegionProxy = getRegion();
    IExViewRepos RepositoryProxy = null;
    SearchFilter filter = new SearchFilter(SearchFilter.NAME +"=RepositoryAgent");
    RepositoryInfo = RegionProxy.listAgents(null, filter);
        if (RepositoryInfo.length != 0) {
```

```
          RepositoryProxy = (IExViewRepos)
ProxyGenerator.newInstance(IExViewRepos.class,
RepositoryInfo[0].getIdentifier().toString(), ProxyGenerator.SYNC);
       } else {log("Cannot locate Repository Agent.");    }
       if (RepositoryProxy == null) { log("No Repository Agent.");  }
       return RepositoryProxy;}
//-------------------------------------------------------------------------//
public boolean MonitorSystemforCollaboratorAvailability(){
   boolean returnIfTargetExists;
   RegionProxy = getRegion();
   IExDiagnosTarget TargetProxy = null;
   SearchFilter filter = new SearchFilter(SearchFilter.DESCRIPTION+"=Target");
   TargetInfo = RegionProxy.listAgents(null,filter);
      if (TargetInfo.length != 0) {
      TargetProxy = (IExDiagnosTarget)
ProxyGenerator.newInstance(IExDiagnosTarget.class, TargetInfo[0].getIdentifier().toString(),
ProxyGenerator.SYNC);
       returnIfTargetExists=true;
       } else { log("Cannot locate Target Agent.");
           returnIfTargetExists=false;      }
       if (TargetProxy == null) {  log("No Target Agent fould.");
          returnIfTargetExists=false; }
return  returnIfTargetExists;}
//-------------------------------------------------------------------------//
boolean TargetSimulatedFailure(){

boolean SuccessOrNot = true;
RegionProxy = getRegion();
IExDiagnosTarget TargetProxy = null;
       SearchFilter filter = new SearchFilter(SearchFilter.DESCRIPTION+"=Target");
       TargetInfo = RegionProxy.listAgents(null,filter);
       if (TargetInfo.length != 0) {
       TargetProxy = (IExDiagnosTarget)
ProxyGenerator.newInstance(IExDiagnosTarget.class, TargetInfo[0].getIdentifier().toString(),
ProxyGenerator.SYNC);
       SuccessOrNot=true;
       } else {log("Cannot locate Target Agent.");
           SuccessOrNot=false; }
       if (TargetProxy == null) {
          log("No Target Agent fould.");
          SuccessOrNot=false;  }
  try {
      TargetProxy.SimulatedTargetFailure();
   }catch (Exception e){e.printStackTrace();   }
return  SuccessOrNot; }
//-------------------------------------------------------------------------//
public void TargetIsGoingDown(AgentInfo TargetInfo){
FailedTarget = TargetInfo;
String []  SimulatedParameters = new String[3];
 SimulatedParameters[0]= "Simulation";
 SimulatedParameters[1]= "Collaborator Unavailable";
```

```
 SimulatedParameters[2]="";
 initialization(SimulatedParameters);}
//-------------------------------------------------------------------------//
boolean MonitorSystemforCorrespondentAvailability(){
   boolean returnIfMasterExists;
   RegionProxy = getRegion();
   IpPerformanceMonitoringRequest MasterProxy = null;
   SearchFilter filter = new SearchFilter(SearchFilter.DESCRIPTION+"=Master");
    MasterInfo = RegionProxy.listAgents(null,filter);
     if (MasterInfo.length != 0) {
       MasterProxy = (IpPerformanceMonitoringRequest)
ProxyGenerator.newInstance(IpPerformanceMonitoringRequest.class,
MasterInfo[0].getIdentifier().toString(), ProxyGenerator.SYNC);
       returnIfMasterExists=true;
       } else { log("Cannot locate Master Agent.");
           returnIfMasterExists=false;       }
       if (MasterProxy == null) {  log("No Master Agent fould.");
          returnIfMasterExists=false;  }
return  returnIfMasterExists; }
//-------------------------------------------------------------------------//
public void DisplayMessageOnMainFailuresGUI (String MessageToDisplay){
   if (GUIReference!=null){
   GUIReference.setStatusAreaText(MessageToDisplay);    }
   else log("The failures GUI is not available.....");}
//-------------------------------------------------------------------------//
/**Causes the falure of the Master agent */
void stopTheMaster (){
   IAgentSystem agencyProxy;
   WorkerEvent MasterProxy = null;
   RegionProxy = getRegion();
   SearchFilter filter = new SearchFilter(SearchFilter.DESCRIPTION+"=Master");
   MasterInfo = RegionProxy.listAgents(null,filter);
      if (MasterInfo.length != 0) {
         MasterProxy = (WorkerEvent) ProxyGenerator.newInstance(WorkerEvent.class,
MasterInfo[0].getIdentifier().toString(), ProxyGenerator.SYNC);
       } else { log("Cannot locate Repository Agent.");       }
       if (MasterProxy == null) {       log("No Repository Agent.");        }
       try{        MasterProxy.KillMaster();        }
       catch(Exception e){        log("Removing Master failed..");      }
}//-------------------------------------------------------------------------//
 public  String [] NetworkNodes(){
   String CurrentLocation = getInfo().getLocation().toString();
   ReadWrite ReadFromFile = new ReadWrite();
   Vector  Servers = ReadFromFile.read("./lib/ParticipatingServers.properties");
   String [] ParticipatingServers= new String [Servers.size()];
         try {
ParticipatingServers[0]="socket://"+(String)Servers.elementAt(1)+":7002/alternative"
ParticipatingServers[1]="socket://"+(String)Servers.elementAt(0)+":7000/destination"
       }       catch (Exception writeErr){ writeErr.printStackTrace(); }
      if (CurrentLocation.equalsIgnoreCase(ParticipatingServers[1]+"/InformationDesk")){
ParticipatingServers[1]="socket://"+(String)Servers.elementAt(1)+":7000/destination"
```

```
return ParticipatingServers; }
//-------------------------------------------------------------------------//
String[] ConvertInputArguments(String Arguments){
   String [] ConvertedArguments = new String[4];
   String s_read = Arguments;
   int index = 0;
      if (Arguments==null){
      DisplayMessageOnMainFailuresGUI("Input arguments have not been provided. Default
values will be used instead!!");
      ConvertedArguments[0]= NetworkNodes()[0];
      ConvertedArguments[1]= NetworkNodes()[1];
      ConvertedArguments[2]="alternative";
      ConvertedArguments[3]="10000";
      }   else {
   StringTokenizer delimitered = new StringTokenizer(s_read,",");
               while (delimitered.hasMoreTokens()){
               ConvertedArguments[index]=delimitered.nextToken();
               System.out.println("ConvertInputArguments are the followings :"
+ConvertedArguments[index]);
               index++;
               }    }

   if
(ConvertedArguments.length<4||ConvertedArguments[0].equalsIgnoreCase("")||ConvertedAr
guments[1].equalsIgnoreCase("")||ConvertedArguments[2].equalsIgnoreCase("")||ConvertedA
rguments[3].equalsIgnoreCase("")){
   DisplayMessageOnMainFailuresGUI("Malformed input arguments. Default values will be
used instead!!");
   ConvertedArguments[0]= NetworkNodes()[0];
   ConvertedArguments[1]= NetworkNodes()[1];
   ConvertedArguments[2]= "alternative";
   ConvertedArguments[3]="10000";    }
 return ConvertedArguments;  }
//-------------------------------------------------------------------------//
String[] ConvertInputArgumentsOperationalResources(String Arguments){
   String [] ConvertedArguments = new String[3];
   String s_read = Arguments;
   int index = 0;
      if (Arguments==null){
      DisplayMessageOnMainFailuresGUI("Input arguments have not been provided. Default
values will be used instead!!");
      ConvertedArguments[0]= NetworkNodes()[0];
      ConvertedArguments[1]= NetworkNodes()[1];
            if (CounterRequestID%2==0){ ConvertedArguments[2]="true";              }
               else{ConvertedArguments[2]="false";      }       }
      else {     StringTokenizer delimitered = new StringTokenizer(s_read,",");
                  while (delimitered.hasMoreTokens()){
                  ConvertedArguments[index]=delimitered.nextToken();
                  System.out.println("ConvertInputArguments are the followings :"
+ConvertedArguments[index]);
                  index++;   }
```

```
      }
      if
(ConvertedArguments.length<3||ConvertedArguments[0].equalsIgnoreCase("")||ConvertedAr
guments[1].equalsIgnoreCase("")||ConvertedArguments[2].equalsIgnoreCase("")){
         DisplayMessageOnMainFailuresGUI("Malformed input arguments. Default values
will be used instead!!");
         ConvertedArguments[0]= NetworkNodes()[0];
         ConvertedArguments[1]= NetworkNodes()[1];
         ConvertedArguments[2]="true";   }
return ConvertedArguments;  }
//-------------------------------------------------------------------------//
public void TerminateMonitors(){
   int index= 0;
      if (ControllerCreated!=null){
         ControllerCreated.trimToSize();
         Object [] ControllerModules= new Object[ControllerCreated.size()];
         ControllerModules=ControllerCreated.toArray();
                  try{ log ("Trying to remove monitors.....");
                    for( index= 0; index<ControllerModules.length;index++){
                           if (((IExControllerNetMang)ControllerModules[index])!=null){

((IExControllerNetMang)ControllerModules[index]).TerminateAdaptationLogicModules();
} }}
                catch (Exception e){
                   log ("I cannot terminate monitors.....");
                   e.printStackTrace();                    }
            log ("I have terminate  : "+index +"  monitors");       }
  ControllerCreated=null;}
//-------------------------------------------------------------------------//
/**
* Gets a reference to Worker Agent ( Interface WorkerRequest ).
* @return WorkerRequest. A reference of the WorkerRequest interface.
*/
WorkerRequest getReferenceToWorker(){
   IRegion WholeRegionProxy =getRegion();
   WorkerRequest WorkerProxy = null;
   SearchFilter filter = new SearchFilter(SearchFilter.DESCRIPTION+"=Worker");
   AgentInfo[] WorkerInfo = WholeRegionProxy.listAgents(null,filter);
      if (WorkerInfo.length != 0) {
         WorkerProxy = (WorkerRequest) ProxyGenerator.newInstance(WorkerRequest.class,
WorkerInfo[0].getIdentifier().toString(), ProxyGenerator.SYNC);
         log.print(Constants.INFO, "I regain reference to Worker");
      } else {    System.out.println("Cannot locate Worker Agent.");          }
      if (WorkerProxy == null) {  System.out.println("No Worker Agent.");       }
 return WorkerProxy;}
//-------------------------------------------------------------------------//
boolean Reaatempt= false;
int ReattemptRequest= 0;
AgentInfo[] MasterInfo;
AgentInfo[] TargetInfo;
AgentInfo FailedTarget;
```

```
static String CurrentStatus;
int CounterRequestID=0;
Object[] CurrentArguments;
public static Object DataTypeTemlate;
/*******************************************************/
}
```

25. View

```
package uk.ac.surrey.callisto.pm.adaptation;
import uk.ac.surrey.callisto.pm.*;
import de.ikv.grasshopper.agent.*;
import de.ikv.grasshopper.agency.*;
import de.ikv.grasshopper.type.*;
import de.ikv.grasshopper.communication.*;
import de.ikv.grasshopper.util.*;
import java.util.*;
import java.net.*;
/** This class specifies the View Module of the Intelligent Adaptation to the
 * Environment Module. It is instanciated by the Adaptation Logic Module in
 * order to support communication of the latter with the Diagnostics, Repository, and Worker
agents. */
public class View implements IInAdaptLogicView{
 /**A reference to the Diagnostics agent provided from the Adaptation logic
 * module */
 IExViewDiagnos ReferenceToDiagnostics;
/** A reference to the Repository agent provided from the Adaptation logic
 * module */
 IExViewRepos ReferenceToRepository;
//-------------------------------------------------------------------------//
public TpAdaptationResult RequestReportFailureCorrespondent(int
RequestID,TpCollaboratorCorrespondentUnavailable Message, IExViewDiagnos
ReferenceToDiagnosticsObject, IExViewRepos ReferenceToRepositoryObject ){
   this.ReferenceToDiagnostics =(IExViewDiagnos)ReferenceToDiagnosticsObject;
   this.ReferenceToRepository =(IExViewRepos)ReferenceToRepositoryObject;
   TpAdaptationResult message =
ReferenceToDiagnostics.IdentifyFailureCorrespondent(RequestID,Message);
   return message;}
//-------------------------------------------------------------------------/
/public TpAdaptationResult ResourceUnavailable(int RequestID,TpResourceUnavailable
Message,IExViewDiagnos ReferenceToDiagnosticsObject, IExViewRepos
ReferenceToRepositoryObject ){
   this.ReferenceToDiagnostics =(IExViewDiagnos)ReferenceToDiagnosticsObject;
   this.ReferenceToRepository =(IExViewRepos)ReferenceToRepositoryObject;
   TpAdaptationResult message =
ReferenceToDiagnostics.IdentifyResourceUnavailable(RequestID,Message);
   return message;    }
//-------------------------------------------------------------------------//
public TpAdaptationResult RequestReportFailureProximity(int
RequestID,TpProximityFailure Message,IExViewDiagnos ReferenceToDiagnosticsObject,
IExViewRepos ReferenceToRepositoryObject ){
```

```
   this.ReferenceToDiagnostics =(IExViewDiagnos)ReferenceToDiagnosticsObject;
   this.ReferenceToRepository =(IExViewRepos)ReferenceToRepositoryObject;
   TpAdaptationResult message =
ReferenceToDiagnostics.IdentifyProximityFailure(RequestID,Message);
   return message;}
//-------------------------------------------------------------------------//
public TpAdaptationResult RequestReportCriticalResourceUtilization(int RequestID,
TpOperationalResourcesUtilization Message, IExViewDiagnos
ReferenceToDiagnosticsObject, IExViewRepos ReferenceToRepositoryObject ){
   this.ReferenceToDiagnostics =(IExViewDiagnos)ReferenceToDiagnosticsObject;
   this.ReferenceToRepository =(IExViewRepos)ReferenceToRepositoryObject;
   TpAdaptationResult message =
ReferenceToDiagnostics.IdentifyCriticalResourceUtilization(RequestID,Message);
   return message;}
//-------------------------------------------------------------------------//
public void MigrateToNode(int RequestID,Object TpMessage,String
MessageType,TpNodeToMigrate Message, IExViewDiagnos ReferenceToDiagnosticsObject,
IExViewRepos ReferenceToRepositoryObject ){
   this.ReferenceToDiagnostics =(IExViewDiagnos)ReferenceToDiagnosticsObject;
   this.ReferenceToRepository =(IExViewRepos)ReferenceToRepositoryObject;
getReferenceToWorkerToMigrate(ReferenceToDiagnostics).MigrateToNode(RequestID,Tp
Message,MessageType, Message);}
//-------------------------------------------------------------------------//
public TpAdaptationResult CreateEntity(int RequestID,TpCreateEntity
Message,IExViewRepos ReferenceToRepositoryObject ){
   this.ReferenceToRepository =(IExViewRepos)ReferenceToRepositoryObject;
   TpAdaptationResult message;
   boolean SuccessfulOrNotResponse =
ReferenceToRepository.CreateEntity(RequestID,Message);
    if (SuccessfulOrNotResponse){
       message = new TpAdaptationResult(new TpAdaptationMessage("True",RequestID,new
TpErrorType("Create entity", "Someone")),true,"OK",Message.Entity, new Object[2]);
    } else {   message = new TpAdaptationResult(new
TpAdaptationMessage("False",RequestID ,new TpErrorType("Create entity",
"Someone")),true,"OK",Message.Entity, new Object[2]); }
   return message; }
//-------------------------------------------------------------------------//
public TpAdaptationResult AdaptationCommand(int RequestID,Object Message){
   TpAdaptationResult message = new TpAdaptationResult(new
TpAdaptationMessage("",1,new TpErrorType("Error", "Error1")),true,"OK","Malakia", new
Object[2]);
   return message;}
//-------------------------------------------------------------------------//
public void ReportStatus(TpStatus Message,IExViewDiagnos ReferenceToDiagnostics){
  IExStatusReportView WorkerRef= null;
             try{          WorkerRef =
getReferenceToWorkerToReportStatus(ReferenceToDiagnostics);
             } catch (Exception e) {     }
             if (WorkerRef!=null){
              try{
                WorkerRef.ReportStatus(Message);
```

```
            } catch (Exception e) {            System.out.println("Failed to pass report status to
Worker Agent.");     }            } }
//-------------------------------------------------------------------------//
public void SetStatus(TpStatus Message,IExViewDiagnos ReferenceToDiagnosticsObject ){
  IExStatusReportView WorkerRef= null;
              try{     WorkerRef =
getReferenceToWorkerToReportStatus(ReferenceToDiagnosticsObject);
            } catch (Exception e) {
               System.out.println("Failed get reference to Worker Agent.");
               e.printStackTrace();            }
            if (WorkerRef!=null){
             try{
               WorkerRef.SetStatus(Message);
            } catch (Exception e) { System.out.println("Failed get pass set status to Worker
Agent");   }}}
//-------------------------------------------------------------------------//
IExStatusReportView getReferenceToWorkerToReportStatus(IExViewDiagnos
ReferenceToDiagnosticsObject){
   IExStatusReportView WorkerProxy = null;
   AgentInfo[] WorkerInfo =ReferenceToDiagnosticsObject.getReferenceToWorker();
      if (WorkerInfo.length != 0) {
         WorkerProxy = (IExStatusReportView)
ProxyGenerator.newInstance(IExStatusReportView.class,
WorkerInfo[0].getIdentifier().toString(), ProxyGenerator.SYNC);
      } else {  System.out.println("Cannot locate Worker Agent.");       }
      if (WorkerProxy == null) {  System.out.println("No Worker Agent.");       }
   return WorkerProxy ;}
//-----------------------------------------------------------------------------------------------//
IExViewCarrier getReferenceToWorkerToMigrate(IExViewDiagnos
ReferenceToDiagnosticsObject){
   IExViewCarrier WorkerProxyCarrier = null;
   AgentInfo[] WorkerInfo =ReferenceToDiagnosticsObject.getReferenceToWorker();
      if (WorkerInfo.length != 0) {
         WorkerProxyCarrier = (IExViewCarrier)
ProxyGenerator.newInstance(IExViewCarrier.class, WorkerInfo[0].getIdentifier().toString(),
ProxyGenerator.SYNC);
      } else {  System.out.println("Cannot locate Worker Agent.");          }
      if (WorkerProxyCarrier == null) { System.out.println("No Worker Agent.");     }
   return WorkerProxyCarrier ;}
//-----------------------------------------------------------------------------------------------//
}
```

26. TargetWithRepositoryAndDiagnostics

```
package uk.ac.surrey.callisto.pm.adaptation;
import uk.ac.surrey.callisto.pm.*;
import uk.ac.surrey.callisto.pm.adaptation.UpdateProperties.ReadWrite;
import de.ikv.grasshopper.agent.*;
import de.ikv.grasshopper.type.*;
import de.ikv.grasshopper.util.*;
```

```
import de.ikv.grasshopper.agency.*;
import de.ikv.grasshopper.communication.*;
import com.adventnet.snmp.beans.*;
import javax.swing.*;
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
import java.util.*;

/**
 * Allows monitors to take passive and active measurements of network performance. Passive
measurements are allowed
 * through the SNMP protocol, while active measurements are supported via the return of an
'echo' string. Target features include:
 *  <ul><li>Open access to any SNMP object by providing a wrapper for snmpget.</li>
 *  <li>Caters for the synchronization needed when two or more monitors may try to access a
Target method at the same time.</li>
 *  <li>A monitor can load additional MIBs as required.</li>
 *  <li>Offers support for reply of an echo string.</li>
 *  <li>Inlcudes a simple GUI used to test its operation.</li></ul>
 */
public class TargetWithRepositoryAndDiagnostics extends StationaryAgent implements
TargetRequest,LocationInfo,IExDiagnosTarget,ISystemListener {
    /* An SNMP target session. */
    private SnmpTarget target;
    /* Logging facilities. */
    private Log log;
    /* The object name. */
    private String  name = "Target";
   /* A reference to the Interface of the agency*/
    IAgentSystem AgencyProxyForRepositoryAndDiagnostics;
    AgentInfo DiagnosticsAgentInfo = null;
    AgentInfo RepositoryAgentInfo = null;
    /** Empty object initialization. */
    public TargetWithRepositoryAndDiagnostics() {    }
//-----------------------------------------------------------------------------------------//
    /** Specifies agent behaviour. The SNMP initializations required take place
     * here. This agent has no other behaviour while in existence. Instead, it
     * waits passively for requests of network information.
     */
    public void live() {
        AgencyProxyForRepositoryAndDiagnostics = getAgentSystem();
        CreateRepositoryAndDiagnosticsAgents();
        // Enable logging.
        log = new Log();
        log.enabled = true;
        log.name = name;
        // Get properties.
        Properties snmp = new Properties();
```

```
        try {
           FileInputStream in = new FileInputStream("lib/snmp.properties");
           snmp.load(in);
           in.close();
           log.print(Constants.INFO, "SNMP properties successfully read from file.");
        } catch (Exception e) {
           log.print(Constants.ERROR, "Failed to read SNMP properties from file.");
           log.print(Constants.DEBUG, e.toString());
        }
        target = new SnmpTarget();
        loadMIB((String)snmp.get("mib_path") + (String)snmp.get("default_mib"));
  synchronized(waitLock) {
    try { waitLock.wait(); // wait for notification   }
    catch(InterruptedException e) {log("interrupted: proceeding");   }
  }  try {  remove();  }
  catch(Exception e) {    log("failed to remove ourselves", e); }    }
//-------------------------------------------------------------------------------------------//
    /**
     * Provide the agent name.
     * @return A name.     */
    public String getName() {
        // Create object name.
        String address = null;
        try {          address = (InetAddress.getLocalHost()).getHostAddress();
        } catch (Exception e) {
           log.print(Constants.ERROR, "Local host address error.");
           log.print(Constants.DEBUG, e.toString());        }
         name = "Target" + address;
        return name;    }
//-------------------------------------------------------------------------------------------//
    /**
     * Provide an agent description.
     * @return the description.
     */
    public String getDescription() {
    return "Target";    }
//-------------------------------------------------------------------------------------------//
 public void beforeRemove() {
    IRegion RegionProxy = getRegion();
    IExTargetFailure WorkerExtentionProxy = null;
    AgentInfo [] WorkerExtentionAgentInfo;
    AgentInfo TargetInfo= getInfo();
           SearchFilter filter = new
SearchFilter(SearchFilter.DESCRIPTION+"=WorkerExtentionAgent");
           WorkerExtentionAgentInfo = RegionProxy.listAgents(null,filter);
           if (WorkerExtentionAgentInfo.length != 0) {
           WorkerExtentionProxy = (IExTargetFailure)
ProxyGenerator.newInstance(IExTargetFailure.class,
WorkerExtentionAgentInfo[0].getIdentifier().toString(), ProxyGenerator.SYNC);
           } else { log("Cannot locate WorkerExtentionAgent.");       }
           if (WorkerExtentionProxy == null) {
```

```
            log("No WorkerExtentionAgent fould.");          }
   try {       WorkerExtentionProxy.TargetIsGoingDown(TargetInfo);    }
   catch (Exception e){    log ("Failed to communicate to the WorkerExtentionAgent");   }
   try {  getReferenceToWorkerExtentionAgent().DisplayMessageOnMainFailuresGUI(
getName()+" has failed");

   }    catch (Exception e){ log ("Failed to communicate to the WorkerExtentionAgent");    }}
//-------------------------------------------------------------------------------------------//
   /* BEGIN: TargetRequest Interface */
   /**
    * Perform an snmpget (Using SNMP v2C).
    * @param oids A list of object IDs.
    * @param targetHost The address of the targeted node.
    * @param community The SNMP community.
    * @param port The port.
    * @param retries Number of retries if failed.
    * @param timeout Timeout in seconds.
    * @return The result.
    */
public synchronized String[] snmpget(String[] oids,
                                     String targetHost,
                                     String community,
                                     String port,
                                     String retries,
                                     String timeout) {

         // Use an SNMP target bean to perform SNMP operations.
      target.setObjectIDList(oids);
      target.setTargetHost(targetHost);
      if (community != null) // Set the community if specified.
         target.setCommunity(community);
      try { // Set the timeout/retries/port parameters, if specified.
         if (port != null)
            target.setTargetPort(Integer.parseInt(port));
         if (retries != null)
            target.setRetries(Integer.parseInt(retries));
         if (timeout != null)
            target.setTimeout(Integer.parseInt(timeout));
      } catch (NumberFormatException e) {
         log.print(Constants.INFO, "An error occured while preparing SNMP session.");
         log.print(Constants.DEBUG, e.toString());
      }
      String result[] = target.snmpGetList();  // do a get request
         if (result == null) {
         log.print(Constants.INFO, "SNMP reqest error.");
         log.print(Constants.DEBUG, target.getErrorString());        }
      return result;
    }
//-------------------------------------------------------------------------------------------//
   /**
    * Load an additional MIB file.
```

```
     * @param The path and filename of the MIB.
     */
public synchronized void loadMIB(String mib) {
        try {
           target.loadMibs(mib);
           log.print(Constants.INFO, mib + " MIB loaded succesfully.");
        } catch (Exception e) {
           log.print(Constants.ERROR, "Failed to load the specified MIB.");
           log.print(Constants.DEBUG, e.toString());
        }    }
//-----------------------------------------------------------------------------------------------//
   /**
     * Allows for active measurements of network performance.
     * @param test The test string.
     * @return The test string echoed.
     */
public synchronized byte[] echo(byte[] test) {
        return test;    }
    /* END: TargetRequest Interface */
//-----------------------------------------------------------------------------------------------//
//BEGIN: LocationInfo Interface
/** Gets the Allovated memory (Total memory - Free memory)*/
public long getAllocMemory()
{    long freeMemory,totalMemory,allocatedMemory;
    freeMemory = Runtime.getRuntime().freeMemory();
    totalMemory = Runtime.getRuntime().totalMemory();
    allocatedMemory = totalMemory - freeMemory;
return allocatedMemory;}
//-----------------------------------------------------------------------------------------------//
/**Gets the number of Agents from each Agency*/
public int getNumberOfAgents()
{    IAgentSystem agencyProxy;
    AgentInfo[] remoteAgents;
    //get proxy of local agency
    agencyProxy = getAgentSystem();
    GrasshopperAddress agencyAddress = getInfo().getLocation();
agencyProxy=(IAgentSystem)ProxyGenerator.newInstance(IAgentSystem.class,agencyAddre
ss.generateAgentSystemId(),agencyAddress);
    remoteAgents = agencyProxy.listAgents(new SearchFilter());
return remoteAgents.length-1; }
//END: LocationInfo Interface
//-----------------------------------------------------------------------------------------------//
 public void CreateRepositoryAndDiagnosticsAgents(){
    try {
    DiagnosticsAgentInfo =
AgencyProxyForRepositoryAndDiagnostics.createAgent("uk.ac.surrey.callisto.pm.adaptation.
DiagnosticsAgent",getInfo().getCodebase(),null,null);    }
    catch (AgentCreationFailedException e){
    log("Failed to create Diagnostics Agent");
    e.printStackTrace();    }
   try {
```

```
    RepositoryAgentInfo =
AgencyProxyForRepositoryAndDiagnostics.createAgent("uk.ac.surrey.callisto.pm.adaptation.
RepositoryAgent",getInfo().getCodebase(),null,null);     }
    catch (AgentCreationFailedException e){
    log("Failed to create Repository Agent");
    e.printStackTrace();    } }
//*****************************************************************//
public TpAdaptationResult ReportFailureCorrespondent(int ReportID,
                                    TpCollaboratorCorrespondentUnavailable Message){
   TpAdaptationResult message = new TpAdaptationResult(new
TpAdaptationMessage("Target
Agent",Message.MessageID,Message.Error),true,"","",checkForResourceAvailability("null_p
lace"));
   return message; }
//-------------------------------------------------------------------------//
public TpAdaptationResult ReportResourceUnavailable(int
RequestID,TpResourceUnavailable Message){
TpAdaptationResult message = new TpAdaptationResult(new TpAdaptationMessage("Target
Agent",Message.MessageID,Message.Error),true,"","",checkForResourceAvailability(Messag
e.ResourceName));
   return message; }
//-------------------------------------------------------------------------//
public TpAdaptationResult ReportProximityFailure(int ReportID, TpProximityFailure
Message){
   TpAdaptationResult message = new TpAdaptationResult(new
   TpAdaptationMessage("Target Agent",Message.MessageID,Message.Error),true,"","",
   checkProximityFailures(Message));
      return message; }
//-------------------------------------------------------------------------//
public TpAdaptationResult ReportCriticalResourceUtilization(int ReportID,
                                        TpOperationalResourcesUtilization Message){

   //TpAdaptationResult message = new TpAdaptationResult(new
TpAdaptationMessage("Target
Agent",Message.MessageID,Message.Error),true,"","",checkForResourceAvailability(Messag
e.NetworkLocation));
   TpAdaptationResult message = new TpAdaptationResult(new
TpAdaptationMessage("Target
Agent",Message.MessageID,Message.Error),true,"","",checkForResourceAvailability("null_p
lace"));
   return message;
}
public void SimulatedTargetFailure(){
  synchronized(waitLock) {
    waitLock.notify();
  }}//-------------------------------------------------------------------------//
public IExPrintStatutsMessages getReferenceToWorkerExtentionAgent(){
    AgentInfo[] WorkerExtentionAgentInfo;
    IRegion RegionProxy = getRegion();
    IExPrintStatutsMessages WorkerExtentionAgentProxy = null;
```

```
        SearchFilter filter = new
SearchFilter(SearchFilter.DESCRIPTION+"=WorkerExtentionAgent");
        WorkerExtentionAgentInfo = RegionProxy.listAgents(null,filter);
        if (WorkerExtentionAgentInfo.length != 0) {
            WorkerExtentionAgentProxy = (IExPrintStatutsMessages)
ProxyGenerator.newInstance(IExPrintStatutsMessages.class,
WorkerExtentionAgentInfo[0].getIdentifier().toString(), ProxyGenerator.SYNC);
        } else {         log("Cannot locate WorkerExtentionAgent.");        }
        if (WorkerExtentionAgentProxy == null) {log("No WorkerExtentionAgent.");  }
 return WorkerExtentionAgentProxy;}
//-------------------------------------------------------------------------//
Object [] checkProximityFailures(TpProximityFailure Message){
    Object [] ProximityFailureInfo= new Object [3];
    ProximityFailureInfo[0]= new Long(getAllocMemory());
    ProximityFailureInfo[1]= new Integer(getNumberOfAgents());
    ProximityFailureInfo[2]= new Boolean(DiagnosProximityFailures (Message));
return ProximityFailureInfo;   }

//-------------------------------------------------------------------------//
boolean DiagnosProximityFailures (TpProximityFailure Message){
  boolean AgencyExists=false;
  String  AgencyToSearch= Message.AgencyName;
  String SearchAgency=null;
  AgentSystemInfo AgencyList[]= null;
    if (AgencyToSearch.equalsIgnoreCase("alternative")){
    SearchAgency= "NAME=alternative";    }
    else{    SearchAgency= "NAME="+AgencyToSearch;    }
    try {
  AgencyList = getRegion().listAgencies(null,new SearchFilter(SearchAgency));    }
    catch (Exception e){    System.out.println("System not found");    }
   if ((AgencyList != null) && (AgencyList.length > 0)) {    AgencyExists= true;   }
   else { System.out.println("The Agency " + AgencyToSearch+" not found ");
        AgencyExists=false;  }
  return AgencyExists;  }

Object [] checkForResourceAvailability( String resourceName){
    Object [] ResourceAvailability= new Object [3];
    ResourceAvailability[0]= new Long(getAllocMemory());
    ResourceAvailability[1]= new Integer(getNumberOfAgents());
    ResourceAvailability[2]= new Boolean(checkPlaceAvailability (resourceName));
return ResourceAvailability;       }
//-------------------------------------------------------------------------//
boolean checkPlaceAvailability (String AgencyToSearch){
  boolean AgencyExists=false;
  String SearchAgency=null;
  AgentSystemInfo AgencyList[]= null;

    if (AgencyToSearch.equalsIgnoreCase("null_place")){
    SearchAgency= "NAME=alternative";
    }    else{  SearchAgency= "NAME="+AgencyToSearch;    }
    AgencyList = getRegion().listAgencies(null,new SearchFilter(SearchAgency));
```

```
        if ((AgencyList != null) && (AgencyList.length > 0)) {
         AgencyList[0].toString();
         AgencyExists= true;      }
  return AgencyExists;  }
//-------------------------------------------------------------------------//
public void agencyAdded(AgentSystemInfo info) {  info.toString();   }
public void agencyRemoved(AgentSystemInfo info) {  info.toString();   }
public void agentAdded(AgentInfo info){ }
public void agentChanged(AgentInfo info) { }
public void agentRemoved(AgentInfo info) { }
public  Identifier getIdentifier() {    return (new Identifier()); }
 public void placeAdded(PlaceInfo info) { }
 public void placeChanged(PlaceInfo info) { }
 public void placeRemoved(PlaceInfo info){ }
//*************************************************************************//
 protected Object waitLock = new Object();}
```

27. RepositoryAgent

```
package uk.ac.surrey.callisto.pm.adaptation;
import uk.ac.surrey.callisto.pm.*;
import uk.ac.surrey.callisto.pm.adaptation.UpdateProperties.*;
import de.ikv.grasshopper.agent.*;
import de.ikv.grasshopper.type.*;
import de.ikv.grasshopper.util.*;
import de.ikv.grasshopper.agency.*;
import de.ikv.grasshopper.communication.*;
import java.util.*;
import java.net.*;
import javax.swing.JOptionPane;
import javax.swing.*;
/** This agent creates the required network management entities and
 *  sends them to the desired network location.*/
public class RepositoryAgent extends StationaryAgent implements IExViewRepos {
     /** The object's name. */
    private String name = null;
   /** Logging facility. */
    Log  log = new Log();
   /*** Counters the entities that have been created before the invoke of the
    * agent is called   */
    int counter =0;
       /** States if the creation of an entity was successful or not   */
    boolean returnStatus;
    /** Creation arguments for the Worker agent.  */
    TpCreationArgumements WorkerInitializationArguments= null;


//-------------------------------------------------------------------------//
  public void live() {}
//-------------------------------------------------------------------------//
```

```
 public void init(Object[] args) {  }
//-------------------------------------------------------------------------//
  public void action() {
    JOptionPane.showMessageDialog(null, "I have created so far  : "+ counter+"  entities",
"Repository Agent",1,null);     }
//-------------------------------------------------------------------------//
  public String getName() { name = "RepositoryAgent";
  return name;}
//-------------------------------------------------------------------------//
public String getDescription(){  return name;}
//-------------------------------------------------------------------------//
//-------------------------------------------------------------------------//
 /** Creates an entity.
 * @param RequestID. The increasing number of the request
 * @param Message. A TpCreateEntity data type that specifies the entity and
 * the network location that this entity will be dispatched */
public boolean CreateEntity(int RequestID, TpCreateEntity Message){
   counter++;
   String CurrentLocation = getInfo().getLocation().toString();
    if (Message.Entity.toString().equalsIgnoreCase("Collaborator")){
    checkPlaceState(Message);
    CreateCreationAgent(Message);     }
   if (Message.Entity.toString().equalsIgnoreCase("Correspondent")){
    checkPlaceState(Message);
    CreateCreationAgent(Message);    }
   if (Message.Entity.toString().equalsIgnoreCase("Logic name")){
     checkPlaceState(Message);
        if (CurrentLocation.equalsIgnoreCase(Message.NetworkLocation) ){
           Create(RequestID,Message);        }
        else {           CreateCreationAgent(Message);         }
   }   return returnStatus;  }
//-------------------------------------------------------------------------//
/**
    * Checks the place and the agency where the entity will be dispatched
    * @param Message. A TpCreateEntity data type that specifies the entity
    */
void checkPlaceState(TpCreateEntity Message){
    GrasshopperAddress locationToMove= new
GrasshopperAddress(Message.NetworkLocation);
    System.out.println("The address is the following      :"+locationToMove.toString());
    System.out.println("The host is the following         :"+locationToMove.getHost());
    System.out.println("The protocol is the following     :"+locationToMove.getProtocol());
    System.out.println("The port is the following         :"+locationToMove.getPort());
    System.out.println("The place is the following        :"+locationToMove.getPlace());
    System.out.println("The agency name is the following
:"+locationToMove.getAgencyName());
    System.out.println("The try other ports is            :"+locationToMove.tryOtherPorts()); }
//-------------------------------------------------------------------------//
   /* Creates the creation agent that will be then transfered to the network
    * location where it will create the desired entity
    * @param Message. A TpCreateEntity data type that specifies the entity and
```

```
   */
void CreateCreationAgent(TpCreateEntity Message){
   int state =0;
   String address = null;
   String agencyAddress = null;
   IAgentSystem agencyProxy = getAgentSystem();
   AgentInfo mobileAgentInfo = null;
   Object [] args = new Object [3];
      try {
         address = (InetAddress.getLocalHost()).getHostAddress();
        } catch (Exception e) {
         log.print(Constants.ERROR, "Local host address error.");
         log.print(Constants.DEBUG, e.toString());        }
   agencyAddress="socket://"+address+":7000/destination";
   if (Message.NetworkLocation.toString().equalsIgnoreCase(agencyAddress)){
   state =1;}
   args[0] = new Integer(state);                // state
   args[1] = Message.NetworkLocation.toString();//location
   args[2] = Message.Entity.toString();// entity
if (Message.Entity.toString().equalsIgnoreCase("Correspondent")){
//String masterAdrs = Message.NetworkLocation.toString().substring(0,24);
//args[1] =masterAdrs+"7000/destination";
args[1] = NetworkNodesMaster();}
  try {      mobileAgentInfo =
agencyProxy.createAgent("uk.ac.surrey.callisto.pm.adaptation.CreationAgent",getInfo().getC
odebase(),null,args);
         returnStatus=true;
       } catch (AgentCreationFailedException e) {
         log("Failed to create Creation Agent.");
         e.printStackTrace();
         returnStatus=false;          }}
//-------------------------------------------------------------------------------------------------/
//**  Creates an entity locally.*/
public void Create(int RequestID, TpCreateEntity Message){
  int index=0;
  String CurrentLocation = getInfo().getLocation().toString();
  String entityToCreate= "Logic";
  IRegion RegionProxy = getRegion();
  AgentInfo [] SearchForAnyLogicComponent;
  AgentInfo AgentInfo = null;
  IAgentSystem agencyProxy = getAgentSystem();
  SearchFilter filter = new SearchFilter(SearchFilter.NAME+"=Logic");
  SearchForAnyLogicComponent = RegionProxy.listAgents(null,filter);
   for(index=0;index<SearchForAnyLogicComponent.length;index++){
         if
CurrentLocation.equalsIgnoreCase(SearchForAnyLogicComponent[index].getLocation().toSt
ring())){
            try {
getAgentSystem().removeAgent(SearchForAnyLogicComponent[index].getIdentifier());
               log("Logic removed with success...");              }
         catch(Exception e)  {    log("Removing Logic failed..");      }        }    }
```

```
   log("Now I create the   :"+entityToCreate);
   try {      AgentInfo =
agencyProxy.createAgent("uk.ac.surrey.callisto.pm.adaptation.Logic",getInfo().getCodebase(
),null,null);      }
     catch (AgentCreationFailedException e) {
      log("Failed to create the entity"+entityToCreate);        } }
//-------------------------------------------------------------------------//
public String NetworkNodesMaster(){
   String CurrentLocation = getInfo().getLocation().toString();
   ReadWrite ReadFromFile = new ReadWrite();
   Vector  Servers = ReadFromFile.read("./lib/ParticipatingServers.properties");
   String [] ParticipatingServers= new String [3];
   String NodeToSendMaster= null;
         try     {
      ParticipatingServers[0]="socket://"+(String)Servers.elementAt(0)+":7002/alternative"
ParticipatingServers[1]="socket://"+(String)Servers.elementAt(0)+":7000/destination"Particip
atingServers[2]="socket://"+(String)Servers.elementAt(1)+":7000/destination"
       catch (Exception writeErr){         writeErr.printStackTrace();      }
      if (CurrentLocation.equalsIgnoreCase(ParticipatingServers[1]+"/InformationDesk")){
       NodeToSendMaster= ParticipatingServers[2];       }
     if (CurrentLocation.equalsIgnoreCase(ParticipatingServers[2]+"/InformationDesk")){
       NodeToSendMaster= ParticipatingServers[1];       }
return NodeToSendMaster; }
//-------------------------------------------------------------------------//
}
```

28. Logic

```
package uk.ac.surrey.callisto.pm.adaptation;
import uk.ac.surrey.callisto.pm.*;
import de.ikv.grasshopper.agent.*;
import de.ikv.grasshopper.agency.*;
import de.ikv.grasshopper.communication.*;
import de.ikv.grasshopper.util.*;
import java.util.*;
/** * This class represents a logic that the IAEM
 * calls its creation.*/
public class Logic extends StationaryAgent implements IExLogic {

   /** The current date and time in string form    */
   String CurrentTime= null;
   /**    An instance of the Calendar class     */
   Calendar rightNow=null;
   /**     The current date and time in Date form     */
   Date CurrentDate=null;
   /**    Log facility   */
   Log log= new Log();
/** Provides the description of this agent
 @return The description of this agent*/
public String getDescription() {
   return "Logic component responsible to provide the current time";    }
```

```
/** Provides the name of this agent
 @return The name of this agent
*/

public String getName() {    return "Logic";    }
/** It is called before the agent remove */
public void  beforeRemove(){    log(getDescription()+ "removed from the execution
environment");}
public void init(Object[] creationArgs) {
   rightNow = Calendar.getInstance();
   CurrentTime=rightNow.getTime().toString();
   CurrentDate=rightNow.getTime();    }
public void live() {
   log(getDescription()+ " has been created");
   log("Time of creation "+CurrentTime);     }
/**Provides the date and time of this agent's instance creation
@return The creation date and time of the logic */
public Date getCreationTime(){         return CurrentDate;     }}
```

29. DiagnosticsAgent

```
package uk.ac.surrey.callisto.pm.adaptation;
import de.ikv.grasshopper.agent.*;
import de.ikv.grasshopper.type.*;
import de.ikv.grasshopper.util.*;
import de.ikv.grasshopper.agency.*;
import de.ikv.grasshopper.communication.*;
import de.ikv.grasshopper.agency.*;
import uk.ac.surrey.callisto.pm.*;
import java.util.*;
import java.net.*;
/** Provides diagnostic services to the Intelligent Adaptation to the Envirnment
* system in order for a failure to be verified nad diagnosted.*/
public class DiagnosticsAgent extends StationaryAgent implements IExViewDiagnos {
 /* The object name. */
   private String name = null;

 /*Simple log facility*/
   Log log = new Log();
//-------------------------------------------------------------------------------------------//
  public void live() {}
//-------------------------------------------------------------------------------------------//
  public void init(Object[] args) {}
//-------------------------------------------------------------------------------------------//
  public String getName() {    name = "DiagnosticsAgent";
    return name;    }
//-------------------------------------------------------------------------------------------//
public String getDescription(){ return name;}
//-------------------------------------------------------------------------------------------//
```

```
public TpAdaptationResult IdentifyFailureCorrespondent(int
ReportID,TpCollaboratorCorrespondentUnavailable Message )  {
IExDiagnosTarget ReferenceToTargetAgent = (IExDiagnosTarget)
getReferenceToTargetAgent();
   TpAdaptationResult message =
ReferenceToTargetAgent.ReportFailureCorrespondent(ReportID,Message);
return message;}
//-------------------------------------------------------------------------------------------//
public TpAdaptationResult IdentifyResourceUnavailable(int
RequestID,TpResourceUnavailable Message){
 IExDiagnosTarget ReferenceToTargetAgent =
(IExDiagnosTarget)getReferenceToTargetAgent();
   TpAdaptationResult message =
ReferenceToTargetAgent.ReportResourceUnavailable(RequestID,Message);
return message;}
//-------------------------------------------------------------------------------------------//
public TpAdaptationResult IdentifyProximityFailure(int ReportID,TpProximityFailure
Message ){
IExDiagnosTarget ReferenceToTargetAgent = (IExDiagnosTarget)
getReferenceToTargetAgent();
   TpAdaptationResult message =
ReferenceToTargetAgent.ReportProximityFailure(ReportID,Message);
return message;}
//-------------------------------------------------------------------------------------------//
public TpAdaptationResult IdentifyCriticalResourceUtilization(int ReportID,
TpOperationalResourcesUtilization Message){
   IExDiagnosTarget ReferenceToTargetAgent = (IExDiagnosTarget)
getReferenceToTargetAgent();
   TpAdaptationResult message =
ReferenceToTargetAgent.ReportCriticalResourceUtilization(ReportID,Message);
return message;}
//-------------------------------------------------------------------------------------------//
public IExDiagnosTarget getReferenceToTargetAgent(){
    AgentInfo[] TargetInfo;
    IRegion WholeRegionProxy = getRegion();
    IExDiagnosTarget TargetProxy = null;
    SearchFilter filter = new SearchFilter(SearchFilter.DESCRIPTION+"=Target");
    TargetInfo = WholeRegionProxy.listAgents(null,filter);
       if (TargetInfo.length != 0) {
          TargetProxy = (IExDiagnosTarget)
ProxyGenerator.newInstance(IExDiagnosTarget.class, TargetInfo[0].getIdentifier().toString(),
ProxyGenerator.SYNC);
       } else {           log("Cannot locate Target Agent.");           }
       if (TargetProxy == null) {           log("No Target Agent.");       }
return TargetProxy;}
//-------------------------------------------------------------------------------------------//
public AgentInfo[] getReferenceToWorker(){
    AgentInfo[] WorkerInfo;
    IRegion WholeRegionProxy = getRegion();
    SearchFilter filter = new SearchFilter(SearchFilter.DESCRIPTION+"=Worker");
    WorkerInfo = WholeRegionProxy.listAgents(null,filter);
```

```
   if (WorkerInfo.length == 0) {       log("Cannot locate Worker Agent.");       }
return WorkerInfo;}
//-------------------------------------------------------------------------------------------//}
```

30. CreationAgent

```
package uk.ac.surrey.callisto.pm.adaptation;
import uk.ac.surrey.callisto.pm.adaptation.*;
import uk.ac.surrey.callisto.pm.*;
import de.ikv.grasshopper.agent.*;
import de.ikv.grasshopper.agency.*;
import de.ikv.grasshopper.type.*;
import de.ikv.grasshopper.communication.*;
import de.ikv.grasshopper.util.*;
import java.util.*;
import java.net.*;
import de.ikv.grasshopper.communication.GrasshopperAddress;
import javax.swing.JOptionPane;
/** This class realizes an agent that moves to a remote agency and creates an
 * entity.After this creation removes its self from the execution environment. */
public class CreationAgent extends SmartAgent {
   /** Provides data state.   */
   int state;
   /** The location to migrate in order to create the desired network entity.    */
   String location;
   /** The desired network entity.   */
   String entity;
   String CreateEntity;
//-------------------------------------------------------------------------------------------
   /**Creation arguments are the location and the entity that has to create localy.
   *@param state : 0 or 1 . O if the address of the location field is where the Repository agent
is located.
   *@param location. A Grasshopper address where the new entity will be created
   *@param entity. The entity to be created.   */
  public void init(Object[] creationArgs) {
   /* state = 0;*/
       Integer stateAsInteger = (Integer)creationArgs[0];
       state = stateAsInteger.intValue();
       System.out.println("The state is "+ state);
       location =(String)creationArgs[1];
       entity= (String)creationArgs[2];    }
//-------------------------------------------------------------------------------------------
   public String getName() {       return "CreationAgent";    }
//-------------------------------------------------------------------------------------------
   public void live() {
     String entityToDisplay = "";
         if(entity.equalsIgnoreCase("Collaborator")){
            entityToDisplay="Target Agent";          }
         else if(entity.equalsIgnoreCase("Correspondent"))      {
         entityToDisplay="Master Agent";          }
                  else if (entity.equalsIgnoreCase("Logic name")){
```

```
                    entityToDisplay="Logic";      }
      switch(state) {
         case 0:     if (location != null) {
                state = 1;
         log("Trying to move to the address " + location);
                try {
                boolean AgencyExists=pingAgency(new GrasshopperAddress(location));
                System.out.println("The agency is  available :"+AgencyExists);
                        if (AgencyExists){   move (new GrasshopperAddress(location));}
                        else{
                          log("Abort migration because agency " +location +" does not exist  "
);
                           try {    this.remove();  }
                              catch (Exception e) {printStackTrace();
                                        log("I failed to remove myself", e);
                                        }    }}
                              catch (Exception e) {  e.printStackTrace();
                                                  log("Migration failed: ", e);
                                                  state = 0;}
            }
            break;
                case 1:
         log("Arrived at destination!");
         JOptionPane.showMessageDialog(null,"Now I create the entity  : "+
entityToDisplay,"Repository Agent: Creation Agent",1,null);
                        if (entity.equalsIgnoreCase("Correspondent")){
                          CreateEntity="Master";}
                        if (entity.equalsIgnoreCase("Collaborator")){
                          CreateEntity= "Target";}
                        if (entity.equalsIgnoreCase("Logic name")){
                          CreateEntity= "Logic";}
                        CreateEntity(CreateEntity);
                        state = 0;
                        log("Trying to remove myself from the agency...");
                        try {this.remove();}
                        catch (Exception e) {log("I failed to remove myself", e);      }
                        break;       }     }
//-------------------------------------------------------------------------------------------
void CreateEntity(String entityToCreate){
   IAgentSystem agencyProxy = getAgentSystem();
   AgentInfo AgentInfo = null;
      if (entityToCreate.equalsIgnoreCase("Master")){
      System.out.println("Now i create the   "+entityToCreate);
      try {      AgentInfo =
agencyProxy.createAgent("uk.ac.surrey.callisto.pm.Master",getInfo().getCodebase(),null,null
);       }
      catch (AgentCreationFailedException e) {  log("Failed to create the
entity"+entityToCreate);             }       }
      if (entityToCreate.equalsIgnoreCase("Target")){
         System.out.println("Now I create the   "+entityToCreate);
```

```
        try {          AgentInfo =
agencyProxy.createAgent("uk.ac.surrey.callisto.pm.Target",getInfo().getCodebase(),null,null)
;          }
        catch (AgentCreationFailedException e) {
              log("Failed to create the entity"+entityToCreate);
              }        }
   if (entityToCreate.equalsIgnoreCase("Logic")){
   int index=0;
   String CurrentLocation = getInfo().getLocation().toString();
   IRegion RegionProxy = getRegion();
   AgentInfo [] SearchForAnyLogicComponent;
   SearchFilter filter = new SearchFilter(SearchFilter.NAME+"=Logic");
   SearchForAnyLogicComponent = RegionProxy.listAgents(null,filter);
   for(index=0;index<SearchForAnyLogicComponent.length;index++){
        if
(CurrentLocation.equalsIgnoreCase(SearchForAnyLogicComponent[index].getLocation().toS
tring())){             try                {

getAgentSystem().removeAgent(SearchForAnyLogicComponent[index].getIdentifier());
              log("Logic removed with success...");              }
        catch(Exception e)       {  log("Removing Logic failed..");      }      }   }
   System.out.println("Now I create the   :"+entityToCreate);
   try {    AgentInfo =
agencyProxy.createAgent("uk.ac.surrey.callisto.pm.adaptation.Logic",getInfo().getCodebase(
),null,null);     }
     catch (AgentCreationFailedException e) {
        log("Failed to create the entity"+entityToCreate);}}}
//-------------------------------------------------------------------------------------------    }
```

31. Controller

```
package uk.ac.surrey.callisto.pm.adaptation;
import uk.ac.surrey.callisto.pm.*;
import java.util.*;
import java.net.*;
/** It is created by the WorkerExtentionAgent for handling the encountered
* failures to the Performance Monitoring System.*/
public class Controller implements IExControllerNetMang {
   String CurrentStatus = null;
   Vector AdaptationLogicModulesCreated=new Vector();;
WorkerExtentionAgent ReferenceToWorkerExtentionAgent;
//-------------------------------------------------------------------------//
public Controller(WorkerExtentionAgent ReferenceToWorkerObject){
ReferenceToWorkerExtentionAgent = ReferenceToWorkerObject;}
//-------------------------------------------------------------------------//
public void StartNewTask(int RequestID,TpTaskType Task){
  System.out.println("-----Worker extention ----StartNewTask----------");}
```

```
//-------------------------------------------------------------------------//
public  void setFailureParameters(Object FailureParameters, String
DataType,IExViewDiagnos ReferenceToDiagnosticsObject, IExViewRepos
ReferenceToRepositoryObject ){
  IExViewDiagnos ReferenceToDiagnostics
=(IExViewDiagnos)ReferenceToDiagnosticsObject;
  IExViewRepos ReferenceToRepository =(IExViewRepos)ReferenceToRepositoryObject;
  if (DataType.equalsIgnoreCase("Correspondent Unavailable")){
      TpCollaboratorCorrespondentUnavailable ReceivedDataType
=(TpCollaboratorCorrespondentUnavailable)FailureParameters;
      IInControllerAdaptLogic ReferenceToAdaptationLogic = new  AdaptationLogic();
ReferenceToAdaptationLogic.CorrespondentUnavailable(ReceivedDataType.MessageID,Rec
eivedDataType,ReferenceToDiagnostics,ReferenceToRepository);
      AdaptationLogicModulesCreated.addElement(ReferenceToAdaptationLogic);}
  if (DataType.equalsIgnoreCase("Collaborator Unavailable")){
      TpCollaboratorCorrespondentUnavailable ReceivedDataType
=(TpCollaboratorCorrespondentUnavailable)FailureParameters;
      IInControllerAdaptLogic ReferenceToAdaptationLogic = new AdaptationLogic();
ReferenceToAdaptationLogic.CollaboratorUnavailable(ReceivedDataType.MessageID,Recei
vedDataType, ReferenceToDiagnostics,ReferenceToRepository);
      AdaptationLogicModulesCreated.addElement(ReferenceToAdaptationLogic);}
  if (DataType.equalsIgnoreCase("Resource Unavailable")){
      TpResourceUnavailable ReceivedDataType
=(TpResourceUnavailable)FailureParameters;
      IInControllerAdaptLogic ReferenceToAdaptationLogic = new  AdaptationLogic();
ReferenceToAdaptationLogic.ResourceUnavailable(ReceivedDataType.MessageID,Received
DataType,ReferenceToDiagnostics,ReferenceToRepository);
      AdaptationLogicModulesCreated.addElement(ReferenceToAdaptationLogic);}
}
  if (DataType.equalsIgnoreCase("Logic Unavailable")){
      TpLogicUnavailable ReceivedDataType =(TpLogicUnavailable)FailureParameters;
      IInControllerAdaptLogic ReferenceToAdaptationLogic = new  AdaptationLogic();
ReferenceToAdaptationLogic.LogicUnavailable(ReceivedDataType.MessageID,ReceivedDat
aType,ReferenceToDiagnostics,ReferenceToRepository);
      AdaptationLogicModulesCreated.addElement(ReferenceToAdaptationLogic);
}   if (DataType.equalsIgnoreCase("Migration Denied")){
      TpProximityFailure ReceivedDataType =(TpProximityFailure)FailureParameters;
      IInControllerAdaptLogic ReferenceToAdaptationLogic = new  AdaptationLogic();
ReferenceToAdaptationLogic.ProximityFailure(ReceivedDataType.MessageID,ReceivedData
Type,ReferenceToDiagnostics,ReferenceToRepository );
      AdaptationLogicModulesCreated.addElement(ReferenceToAdaptationLogic);
}   if (DataType.equalsIgnoreCase("RunTime Environment Unavailable")){
      TpProximityFailure ReceivedDataType =(TpProximityFailure)FailureParameters;
      IInControllerAdaptLogic ReferenceToAdaptationLogic = new  AdaptationLogic();
ReferenceToAdaptationLogic.ProximityFailure(ReceivedDataType.MessageID,ReceivedData
Type,ReferenceToDiagnostics,ReferenceToRepository);
      AdaptationLogicModulesCreated.addElement(ReferenceToAdaptationLogic);
}   if (DataType.equalsIgnoreCase("Network Node Inaccessible")){

      TpProximityFailure ReceivedDataType =(TpProximityFailure)FailureParameters;
      IInControllerAdaptLogic ReferenceToAdaptationLogic = new  AdaptationLogic();
```

```
ReferenceToAdaptationLogic.ProximityFailure(ReceivedDataType.MessageID,ReceivedData
Type,ReferenceToDiagnostics,ReferenceToRepository);
        AdaptationLogicModulesCreated.addElement(ReferenceToAdaptationLogic);
}
    if (DataType.equalsIgnoreCase("Remote Communication Failure")){
        TpProximityFailure ReceivedDataType =(TpProximityFailure)FailureParameters;
        IInControllerAdaptLogic ReferenceToAdaptationLogic = new  AdaptationLogic();
ReferenceToAdaptationLogic.ProximityFailure(ReceivedDataType.MessageID,ReceivedData
Type,ReferenceToDiagnostics,ReferenceToRepository);
        AdaptationLogicModulesCreated.addElement(ReferenceToAdaptationLogic);}
   if (DataType.equalsIgnoreCase("Logic Update Required")){
        TpLogicUnavailable ReceivedDataType =(TpLogicUnavailable)FailureParameters;
        IInControllerAdaptLogic ReferenceToAdaptationLogic = new  AdaptationLogic();
ReferenceToAdaptationLogic.LogicUnavailable(ReceivedDataType.MessageID,ReceivedDat
aType,ReferenceToDiagnostics,ReferenceToRepository);
        AdaptationLogicModulesCreated.addElement(ReferenceToAdaptationLogic);
}
   if (DataType.equalsIgnoreCase("Critical Resource Utilization")){
        TpOperationalResourcesUtilization ReceivedDataType
=(TpOperationalResourcesUtilization)FailureParameters;
        IInControllerAdaptLogic ReferenceToAdaptationLogic = new  AdaptationLogic();
ReferenceToAdaptationLogic.OperationalResourcesUnavailability(ReceivedDataType.Messa
geID,ReceivedDataType,ReferenceToDiagnostics,ReferenceToRepository);
        AdaptationLogicModulesCreated.addElement(ReferenceToAdaptationLogic);
}}//-------------------------------------------------------------------------//
 public Object getStatus(){
    Object ObjectName = ReferenceToWorkerExtentionAgent.getStatus();
    return ObjectName; }
//-------------------------------------------------------------------------//
 public void setStatus(String Status){    CurrentStatus = Status; }
//-------------------------------------------------------------------------//
public void TerminateAdaptationLogicModules(){
    AdaptationLogicModulesCreated.trimToSize();
    Object [] AdaptationModules= new Object[AdaptationLogicModulesCreated.size()];
    AdaptationModules=AdaptationLogicModulesCreated.toArray();
        for( int index= 0; index<AdaptationModules.length;index++){
            if (((AdaptationLogic)AdaptationModules[index])!=null){
        ((AdaptationLogic)AdaptationModules[index]).terminate();
        System.out.println("Terminating active monitors");
        }}}//-------------------------------------------------------------------------//}
```

32. AdaptationLogic

```
package uk.ac.surrey.callisto.pm.adaptation;
import uk.ac.surrey.callisto.pm.*;
import java.util.*;
import java.net.*;
import java.io.*;
import java.lang.reflect.*;
import java.lang.*;
/*** The IAEM initiating and controlling a number of monitoring tasks.  */
```

```
public class AdaptationLogic implements IInControllerAdaptLogic{
    public int MethodNumber = 0;
    public boolean terminate=false;
    IInAdaptLogicView ReferenceToView =null;

//-------------------------------------------------------------------------------------------------//
public void CorrespondentUnavailable(int RequestID,
                                TpCollaboratorCorrespondentUnavailable Message,
                                IExViewDiagnos ReferenceToDiagnosticsObject,
                                IExViewRepos ReferenceToRepositoryObject ){
   IInAdaptLogicView ReferenceToView =null;
   MethodNumber = 1;
   IExViewDiagnos ReferenceToDiagnostics
=(IExViewDiagnos)ReferenceToDiagnosticsObject;
   IExViewRepos ReferenceToRepository =(IExViewRepos)ReferenceToRepositoryObject;
   ReferenceToView = new View();
   TpAdaptationResult message =
ReferenceToView.RequestReportFailureCorrespondent(RequestID,Message,ReferenceToDia
gnostics,ReferenceToRepository);
  if (message.ID.equalsIgnoreCase("Target Agent")){
   message = ReferenceToView.CreateEntity(RequestID,new
TpCreateEntity("Correspondent",Message.NetworkLocation),ReferenceToRepository);
       if (message.ID.equalsIgnoreCase("True")){
          System.out.println("Correspondent has been created : "+ message.ID.toString() );
}
       else {           System.out.println("Correspondent has been created : "+
message.ID.toString() );       }  }
  ReferenceToView=null;}
//-------------------------------------------------------------------------------------------------//
public void CollaboratorUnavailable(int RequestID,
                               TpCollaboratorCorrespondentUnavailable Message,
                               IExViewDiagnos ReferenceToDiagnosticsObject,
                               IExViewRepos ReferenceToRepositoryObject ){
   IInAdaptLogicView ReferenceToViewCU =null;
   MethodNumber = 2;
   IExViewDiagnos ReferenceToDiagnostics
=(IExViewDiagnos)ReferenceToDiagnosticsObject;
   IExViewRepos ReferenceToRepository =(IExViewRepos)ReferenceToRepositoryObject;
   ReferenceToViewCU = new View();
   TpAdaptationResult message = ReferenceToViewCU.CreateEntity(RequestID,new
TpCreateEntity("Collaborator", Message.NetworkLocation),ReferenceToRepository);
   ReferenceToViewCU=null;}
//-------------------------------------------------------------------------------------------------//
public void ResourceUnavailable(int RequestID,
                           TpResourceUnavailable Message,
                           IExViewDiagnos ReferenceToDiagnosticsObject,
                           IExViewRepos ReferenceToRepositoryObject ){
   MethodNumber = 3;
   IInAdaptLogicView ReferenceToViewRU =null;
   boolean ReAttemptWillFollow= false;
```

```
   IExViewDiagnos ReferenceToDiagnostics
=(IExViewDiagnos)ReferenceToDiagnosticsObject;
   IExViewRepos ReferenceToRepository =(IExViewRepos)ReferenceToRepositoryObject;
   ReferenceToViewRU = new View();
   TpAdaptationResult message =
ReferenceToViewRU.ResourceUnavailable(RequestID,Message,ReferenceToDiagnostics,Ref
erenceToRepository);
    System.out.println("The return message is :  " +message.AdaptationData[2].toString());

    if (message.AdaptationData[2].toString().equalsIgnoreCase("false")){
    ReAttemptWillFollow= true;
Reattempt(RequestID,Message,"TpResourceUnavailable",ReferenceToDiagnosticsObject,Ref
erenceToRepositoryObject,10000);
    ReferenceToViewRU.ReportStatus(new TpStatus(new
TpAdaptationMessage(Message.ID,Message.MessageID,Message.Error),"The resource was
found
:"+message.AdaptationData[2].toString(),ReAttemptWillFollow,1000),ReferenceToDiagnosti
cs);    }
             if (ReAttemptWillFollow==false) {
              ReferenceToViewRU.SetStatus(new TpStatus(new
TpAdaptationMessage(Message.ID,Message.MessageID,Message.Error),"The resource was
found
:"+message.AdaptationData[2].toString(),ReAttemptWillFollow,1000),ReferenceToDiagnosti
cs);             }
    ReferenceToViewRU =null;}
//-------------------------------------------------------------------------------------------//
public void LogicUnavailable(int RequestID,
                        TpLogicUnavailable Message,
                        IExViewDiagnos ReferenceToDiagnosticsObject,
                        IExViewRepos ReferenceToRepositoryObject ){
   MethodNumber = 4;
   IExViewDiagnos ReferenceToDiagnostics
=(IExViewDiagnos)ReferenceToDiagnosticsObject;
   IExViewRepos ReferenceToRepository =(IExViewRepos)ReferenceToRepositoryObject;
   ReferenceToView = new View();
   TpAdaptationResult message = ReferenceToView.CreateEntity(RequestID,new
TpCreateEntity("Logic name", Message.NetworkLocation),ReferenceToRepository);
   ReferenceToView =null;
  if (Message.ID.equalsIgnoreCase("Logic Unavailable")){
   IInAdaptLogicView ReferenceToViewToSetStatus = new View();
   ReferenceToViewToSetStatus.SetStatus(new TpStatus(new
TpAdaptationMessage(Message.ID,Message.MessageID,Message.Error),"The Logic in the
network node "+Message.NetworkLocation + " has been created
",false,(long)0),ReferenceToDiagnostics);
   ReferenceToViewToSetStatus= null;   }
  if (Message.ID.equalsIgnoreCase("Logic Update Required")){
   IInAdaptLogicView ReferenceToViewToSetStatus = new View();
   ReferenceToViewToSetStatus.SetStatus(new TpStatus(new
TpAdaptationMessage(Message.ID,Message.MessageID,Message.Error),"The Logic in the
network node "+Message.NetworkLocation + " has been updated successfully
",false,(long)0),ReferenceToDiagnostics);
```

```
  ReferenceToViewToSetStatus= null;   }}
//-------------------------------------------------------------------------------------------------//
public void ProximityFailure(int RequestID,
                      TpProximityFailure Message,
                      IExViewDiagnos ReferenceToDiagnosticsObject,
                      IExViewRepos ReferenceToRepositoryObject ){

  MethodNumber = 5;
  boolean ReAttemptWillFollow= false;
  IExViewDiagnos ReferenceToDiagnostics
=(IExViewDiagnos)ReferenceToDiagnosticsObject;
  IExViewRepos ReferenceToRepository =(IExViewRepos)ReferenceToRepositoryObject;
  ReferenceToView = new View();
  if (Message.ID.equalsIgnoreCase("Network Node Inaccessible")){
  TpAdaptationResult message =
ReferenceToView.RequestReportFailureProximity(RequestID,
Message,ReferenceToDiagnostics,ReferenceToRepository);
               if (message.AdaptationData[2].toString().equalsIgnoreCase("false")){
                  ReAttemptWillFollow= true;
Reattempt(RequestID,Message,"TpProximityFailure",ReferenceToDiagnosticsObject,Referen
ceToRepositoryObject,(long)Message.ReattemptPeriod);                    }
               if (message.AdaptationData[2].toString().equalsIgnoreCase("false")){
                  IInAdaptLogicView ReferenceToViewToReport = new View();
                  ReferenceToViewToReport.ReportStatus(new TpStatus(new
TpAdaptationMessage(Message.ID,Message.MessageID,Message.Error),"The network node
"+Message.NetworkLocation + " is
inaccessible",ReAttemptWillFollow,(long)Message.ReattemptPeriod),ReferenceToDiagnostic
s);
                  ReferenceToViewToReport= null;                    }

               if (ReAttemptWillFollow==false)  {
                  ReferenceToView.SetStatus(new TpStatus(new
TpAdaptationMessage(Message.ID,Message.MessageID,Message.Error),"The network node
"+Message.NetworkLocation + " is
inaccessible",ReAttemptWillFollow,(long)Message.ReattemptPeriod),ReferenceToDiagnostic
s);           }   }
  if (Message.ID.equalsIgnoreCase("Remote Communication Failure")){
  TpAdaptationResult message =
ReferenceToView.RequestReportFailureProximity(RequestID,
Message,ReferenceToDiagnostics,ReferenceToRepository);
                  if (message.AdaptationData[2].toString().equalsIgnoreCase("false")){
                     ReAttemptWillFollow= true;
Reattempt(RequestID,Message,"TpProximityFailure",ReferenceToDiagnosticsObject,Referen
ceToRepositoryObject,(long)Message.ReattemptPeriod);                    }
                  if (message.AdaptationData[2].toString().equalsIgnoreCase("false")){
                     IInAdaptLogicView ReferenceToViewToReport = new View();
                     ReferenceToViewToReport.ReportStatus(new TpStatus(new
TpAdaptationMessage(Message.ID,Message.MessageID,Message.Error),"Remote
Communication Failure "+Message.NetworkLocation
,ReAttemptWillFollow,(long)Message.ReattemptPeriod),ReferenceToDiagnostics);
                     ReferenceToViewToReport=null;                    }
```

```
                if (ReAttemptWillFollow==false)                    {
                ReferenceToView.SetStatus(new TpStatus(new
TpAdaptationMessage(Message.ID,Message.MessageID,Message.Error),"Remote
Communication Failure "+Message.NetworkLocation
,ReAttemptWillFollow,(long)Message.ReattemptPeriod),ReferenceToDiagnostics);
 ReferenceToView =null;   }
 if (Message.ID.equalsIgnoreCase("RunTime Environment Unavailable")){
      ReferenceToView = new View();
      TpAdaptationResult message =
ReferenceToView.RequestReportFailureProximity(RequestID,
Message,ReferenceToDiagnostics,ReferenceToRepository);
                if (Message.MigrationPerformed==false&&
message.AdaptationData[2].toString().equalsIgnoreCase("false")){
ReferenceToView.MigrateToNode(RequestID,Message,Message.ID,new
TpNodeToMigrate(Message.NetworkLocation,Message.AlternativeNode,0),ReferenceToDiag
nostics,ReferenceToRepository);                    }
               if (Message.MigrationPerformed==false&&
message.AdaptationData[2].toString().equalsIgnoreCase("true")){
ReferenceToView.MigrateToNode(RequestID,Message,Message.ID,new
TpNodeToMigrate(Message.NetworkLocation,Message.AlternativeNode,1),ReferenceToDiag
nostics,ReferenceToRepository);    }
               if (Message.MigrationPerformed==true&&
message.AdaptationData[2].toString().equalsIgnoreCase("true")){
ReferenceToView.MigrateToNode(RequestID,Message,Message.ID,new
TpNodeToMigrate(Message.NetworkLocation,Message.AlternativeNode,1),ReferenceToDiag
nostics,ReferenceToRepository);                    }
                if
(message.AdaptationData[2].toString().equalsIgnoreCase("false")&&(Message.MigrationPerf
ormed==true)){
                    ReAttemptWillFollow= true;
                    terminate= false;
Reattempt(RequestID,Message,"TpProximityFailure",ReferenceToDiagnosticsObject,Referen
ceToRepositoryObject,(long)Message.ReattemptPeriod);                    }
                if
(message.AdaptationData[2].toString().equalsIgnoreCase("false")&&(Message.MigrationPerf
ormed==true)){
                    IInAdaptLogicView ReferenceToViewToReport = new View();
                    ReferenceToViewToReport.ReportStatus(new TpStatus(new
TpAdaptationMessage(Message.ID,Message.MessageID,Message.Error),"RunTime
Environment Unavailable
"+Message.NetworkLocation,ReAttemptWillFollow,(long)Message.ReattemptPeriod),Refere
nceToDiagnostics);
                    ReferenceToViewToReport= null;
                  }
                if (ReAttemptWillFollow==false)
                  {      ReferenceToView.SetStatus(new TpStatus(new
TpAdaptationMessage(Message.ID,Message.MessageID,Message.Error),"RunTime
Environment Unavailable
"+Message.NetworkLocation,ReAttemptWillFollow,(long)Message.ReattemptPeriod),Refere
nceToDiagnostics);                }
    ReferenceToView = null;  }
```

```
if (Message.ID.equalsIgnoreCase("Migration Denied")){
  ReferenceToView = new View();
  TpAdaptationResult message =
ReferenceToView.RequestReportFailureProximity(RequestID,
Message,ReferenceToDiagnostics,ReferenceToRepository);
                if (Message.MigrationPerformed==false&&
message.AdaptationData[2].toString().equalsIgnoreCase("false")){
ReferenceToView.MigrateToNode(RequestID,Message,Message.ID,new
TpNodeToMigrate(Message.NetworkLocation,Message.AlternativeNode,0),ReferenceToDiag
nostics,ReferenceToRepository);      }
                if (Message.MigrationPerformed==false&&
message.AdaptationData[2].toString().equalsIgnoreCase("true")){
ReferenceToView.MigrateToNode(RequestID,Message,Message.ID,new
TpNodeToMigrate(Message.NetworkLocation,Message.AlternativeNode,1),ReferenceToDiag
nostics,ReferenceToRepository);                    }
                if (Message.MigrationPerformed==true&&
message.AdaptationData[2].toString().equalsIgnoreCase("true")){
ReferenceToView.MigrateToNode(RequestID,Message,Message.ID,new
TpNodeToMigrate(Message.NetworkLocation,Message.AlternativeNode,1),ReferenceToDiag
nostics,ReferenceToRepository);                    }
                if
(message.AdaptationData[2].toString().equalsIgnoreCase("false")&&(Message.MigrationPerf
ormed==true)){
                    ReAttemptWillFollow= true;
                    System.out.println("This ReAttemptWillFollow is
:"+ReAttemptWillFollow);                    terminate= false;
Reattempt(RequestID,Message,"TpProximityFailure",ReferenceToDiagnosticsObject,Referen
ceToRepositoryObject,(long)Message.ReattemptPeriod);        }
                if
(message.AdaptationData[2].toString().equalsIgnoreCase("false")&&(Message.MigrationPerf
ormed==true))  {
                    IInAdaptLogicView ReferenceToViewToReport = new View();
                    ReferenceToViewToReport.ReportStatus(new TpStatus(new
TpAdaptationMessage(Message.ID,Message.MessageID,Message.Error),"Migration Denied
"+Message.NetworkLocation,ReAttemptWillFollow,(long)Message.ReattemptPeriod),Refere
nceToDiagnostics);     ReferenceToViewToReport= null;                }
                if (ReAttemptWillFollow==false)  {
                  ReferenceToView.SetStatus(new TpStatus(new
TpAdaptationMessage(Message.ID,Message.MessageID,Message.Error),"Migration Denied
"+Message.NetworkLocation,ReAttemptWillFollow,(long)Message.ReattemptPeriod),Refere
nceToDiagnostics);}
  ReferenceToView = null;    } }
//-------------------------------------------------------------------------------------------//
public void OperationalResourcesUnavailability(int RequestID,
                                TpOperationalResourcesUtilization Message,
                                IExViewDiagnos ReferenceToDiagnosticsObject,
                                IExViewRepos ReferenceToRepositoryObject ){

  MethodNumber = 6;
  IExViewDiagnos ReferenceToDiagnostics
=(IExViewDiagnos)ReferenceToDiagnosticsObject;
```

```
   IExViewRepos ReferenceToRepository =(IExViewRepos)ReferenceToRepositoryObject;
   ReferenceToView = new View();
   TpAdaptationResult message =
ReferenceToView.RequestReportCriticalResourceUtilization(RequestID,Message,ReferenceT
oDiagnostics,ReferenceToRepository);
  if (message.ID.equalsIgnoreCase("Target Agent")){
    if
(message.AdaptationData[2].toString().equalsIgnoreCase("false")&&Message.ReleaseResour
ce==false){
       ReferenceToView.MigrateToNode(RequestID,Message,Message.ID,new
TpNodeToMigrate(Message.NetworkLocation,Message.AlternativeNode,0),ReferenceToDiag
nostics,ReferenceToRepository);
       IInAdaptLogicView ReferenceToViewToReport = new View();
       ReferenceToViewToReport.ReportStatus(new TpStatus(new
TpAdaptationMessage(Message.ID,Message.MessageID,Message.Error),"Critical Resource
Utilization"+Message.NetworkLocation,false,0),ReferenceToDiagnostics);
       ReferenceToViewToReport= null;    }
     if
(message.AdaptationData[2].toString().equalsIgnoreCase("true")&&Message.ReleaseResourc
e==false){
       ReferenceToView.MigrateToNode(RequestID,Message,Message.ID,new
TpNodeToMigrate(Message.NetworkLocation,Message.AlternativeNode,1),ReferenceToDiag
nostics,ReferenceToRepository);
       IInAdaptLogicView ReferenceToViewToReport = new View();
       ReferenceToViewToReport.ReportStatus(new TpStatus(new
TpAdaptationMessage(Message.ID,Message.MessageID,Message.Error),"Critical Resource
Utilization"+Message.NetworkLocation,false,0),ReferenceToDiagnostics);
       ReferenceToViewToReport= null;    }    if
(message.AdaptationData[2].toString().equalsIgnoreCase("false")&&Message.ReleaseResour
ce==true){
       System.out.println("ADAPTATION LOGIC| Releasing some resources");
       IInAdaptLogicView ReferenceToViewToSetStatus = new View();
       ReferenceToViewToSetStatus.SetStatus(new TpStatus(new
TpAdaptationMessage(Message.ID,Message.MessageID,Message.Error),"Critical Resource
Utilization"+Message.NetworkLocation,true,0),ReferenceToDiagnostics);
       ReferenceToViewToSetStatus= null;      }  }
  System.out.println("ADAPTATION LOGIC| OperationalResourcesUnavailability and
method number is "+ MethodNumber);
  ReferenceToView = null;}
//-------------------------------------------------------------------------------------------------//
public void AdaptationCommand(int RequestID,Object Message){
   MethodNumber = 7;
   System.out.println("-----Adaptation Logic ----- AdaptationCommand method----------");
   System.out.println("RequestID    " +RequestID);
   TpAdaptationResult message = new TpAdaptationResult(new
TpAdaptationMessage("",1,new TpErrorType("Error", "Error1")),true,"OK","Malakia", new
Object[2]);
   System.out.println("ADAPTATION LOGIC| AdaptationCommand and method number is "
+ MethodNumber);
}
//-------------------------------------------------------------------------------------------------//
```

```
public void Reattempt(int RequestID,
                Object Message,
                String MessageType,
                IExViewDiagnos ReferenceToDiagnosticsObject,
                IExViewRepos ReferenceToRepositoryObject,
                long ReattemptPeriod){
ReAttemptClass ReAttemp = new
ReAttemptClass(RequestID,Message,MessageType,ReferenceToDiagnosticsObject,Referenc
eToRepositoryObject,ReattemptPeriod,this,MethodNumber);
ReAttemp.start();
  }
public void terminate(){
 terminate=true;
    System.out.println("ADAPTATION LOGIC : Terminating active monitors....."); }
//*********************************************************************//
class ReAttemptClass implements Runnable {
   int RequestID;
   Object Message;
   String MessageType;
   long ReattemptPeriod;
   AdaptationLogic AdaptationLogicReference;
   int MethodToInvoke;
   IExViewDiagnos ReferenceToDiagnosticsObject;
   IExViewRepos ReferenceToRepositoryObject;
   Thread thread;
   String[] AdaptationLogicMethod ={"",
                        "CorrespondentUnavailable",
                        "CollaboratorUnavailable",
                        "ResourceUnavailable",
                        "LogicUnavailable",
                        "ProximityFailure",
                        "OperationalResourcesUnavailability",
                        "AdaptationCommand"};
//-----------------------------------------------------------------------------------------------//
 public ReAttemptClass(int RequestID,
                Object Message,
                String MessageType,
                IExViewDiagnos ReferenceToDiagnosticsObject,
                IExViewRepos ReferenceToRepositoryObject,
                long ReattemptPeriod,
                AdaptationLogic AdaptationLogicReference,
                int MethodToInvoke){

   this.RequestID=RequestID;
   this.Message=Message;
   this.MessageType=MessageType;
   this.ReferenceToDiagnosticsObject=ReferenceToDiagnosticsObject;
   this.ReferenceToRepositoryObject= ReferenceToRepositoryObject;
   this.ReattemptPeriod=ReattemptPeriod;
   this.AdaptationLogicReference= AdaptationLogicReference;
   this.MethodToInvoke= MethodToInvoke;}
```

```
//-------------------------------------------------------------------------------------------------//
public void start() {
      thread = new Thread(this,"ReAttempt");
      thread.start();}
//-------------------------------------------------------------------------------------------------//
public void run() {
     if (terminate==false){
        try {     thread.sleep(ReattemptPeriod);   // Periodical check.
           switch (MethodToInvoke){
              case 1 :
                    break;
              case 2 :
                    break;
              case 3 :
                    AdaptationLogicReference.ResourceUnavailable(RequestID,
                                                (TpResourceUnavailable)Message,
                                                ReferenceToDiagnosticsObject,
                                                ReferenceToRepositoryObject);
                 break;
             case 4 :   break;
             case 5 :
AdaptationLogicReference.ProximityFailure(RequestID,(TpProximityFailure)Message,Refer
enceToDiagnosticsObject, ReferenceToRepositoryObject);
                 break;            }   } catch (Exception e) {e.printStackTrace();   }       }
  else { System.out.println("Termination of reattempt method has been requested");} }
}
//*******************************************************//}
```

33. MainFauluresGUI

```
package uk.ac.surrey.callisto.pm.adaptation.FailuresGUI;
import java.awt.*;
import uk.ac.surrey.callisto.pm.adaptation.UpdateProperties.*;
import uk.ac.surrey.callisto.pm.adaptation.*;
import uk.ac.surrey.callisto.pm.*;
import javax.swing.*;
/**This class is used in Windows environments**/
public class MainFailuresGUI extends Frame{
        public MainFailuresGUI(){
                setTitle("Main Failures Graphical User Interface");
                setResizable(false);
                setLayout(null);
                setBackground(java.awt.Color.lightGray);
                setForeground(java.awt.Color.black);
                setFont(new Font("Dialog", Font.PLAIN, 12));
                setSize(434,484);
                setVisible(false);
                StartButton.setLabel("Start Emulation");
                add(StartButton);
                StartButton.setBackground(java.awt.Color.lightGray);
                StartButton.setFont(new Font("SansSerif", Font.PLAIN, 12));
```

```
                StartButton.setBounds(54,216,100,24);
                StopButton.setLabel("Clear");
                add(StopButton);
                StopButton.setBackground(java.awt.Color.lightGray);
                StopButton.setFont(new Font("SansSerif", Font.PLAIN, 12));
                StopButton.setBounds(167,216,100,24);
                FailureSelection.addItem("Displays the simulated failures");
                try {FailureSelection.select(0);                    }
                catch (IllegalArgumentException e) { }
                FailureSelection.setFont(new Font("SansSerif", Font.PLAIN, 12));
                add(FailureSelection);
                FailureSelection.setBounds(50,100,339,21);
                CategorySelection.addItem("Availability of Correspondent");
                CategorySelection.addItem("Resource Availability");
                CategorySelection.addItem("Availability of Collaborator");
                CategorySelection.addItem("Availability of Logic");
                CategorySelection.addItem("Proximity failures");
                CategorySelection.addItem("Availability of Operational Resources");
                CategorySelection.addItem("Show all failures");
                try {CategorySelection.select(0);}
                catch (IllegalArgumentException e) { }
                CategorySelection.setFont(new Font("SansSerif", Font.PLAIN, 12));
                add(CategorySelection);
                CategorySelection.setBounds(50,40,339,21);
                SelectCategory.setText("Select failure category:");
                add(SelectCategory);
                SelectCategory.setFont(new Font("SansSerif", Font.PLAIN, 12));
                SelectCategory.setBounds(50,18,339,24);
                SelectFailure.setText("Select failure:");
                add(SelectFailure);
                SelectFailure.setFont(new Font("SansSerif", Font.PLAIN, 12));
                SelectFailure.setBounds(50,78,339,24);
                Status.setText("Status :");
                add(Status);
                Status.setFont(new Font("SansSerif", Font.PLAIN, 12));
                Status.setBounds(50,252,339,25);
                StatusTextArea.setText("In this area messages during the emulation are
displayed ");
        add(StatusTextArea);
                StatusTextArea.setFont(new Font("SansSerif", Font.PLAIN, 12));
                StatusTextArea.setBounds(50,276,339,178);
                LabelArguments.setText("Arguments:");
                add(LabelArguments);
                LabelArguments.setFont(new Font("SansSerif", Font.PLAIN, 12));
                LabelArguments.setBounds(50,132,339,24);
                add(ArgumentsTextField);
                ArgumentsTextField.setBounds(50,156,339,36);
                ResetButton.setLabel("Reset choice");
                add(ResetButton);
                ResetButton.setBackground(java.awt.Color.lightGray);
                ResetButton.setFont(new Font("SansSerif", Font.PLAIN, 12));
```

```
                ResetButton.setBounds(280,216,100,24);
                menu1.setLabel("File");
                menu1.add(aboutMenuItem);
                aboutMenuItem.setLabel("About...");
                menu1.add(menuItemSetupEnvironment);
                menuItemSetupEnvironment.setLabel("Setup Environment");
                menu1.add(exitMenuItem);
                exitMenuItem.setLabel("Exit");
                mainMenuBar.add(menu1);
                setMenuBar(mainMenuBar);
                SymWindow aSymWindow = new SymWindow();
                this.addWindowListener(aSymWindow);
                SymAction lSymAction = new SymAction();
                exitMenuItem.addActionListener(lSymAction);
                aboutMenuItem.addActionListener(lSymAction);
                ResetButton.addActionListener(lSymAction);
                SymItem lSymItem = new SymItem();
                CategorySelection.addItemListener(lSymItem);
                FailureSelection.addItemListener(lSymItem);
                menuItemSetupEnvironment.addActionListener(lSymAction);
                StartButton.addActionListener(lSymAction);
                StopButton.addActionListener(lSymAction);}
public MainFailuresGUI(String title) {
        this();
        setTitle(title);  }
public MainFailuresGUI(WorkerExtentionAgent WorkerExtentionAgentObjectReference)
        {
  this();
   WorkerExtentionAgentReference =WorkerExtentionAgentObjectReference;}
//--------------------------------------------------------------------------//
    /**
     * Shows or hides the component depending on the boolean flag b.
     * @param b  if true, show the component; otherwise, hide the component.
     * @see java.awt.Component#isVisible
     */
    public void setVisible(boolean b)   {
if(b)     {setLocation(50, 50);  }
                super.setVisible(b);     }
//--------------------------------------------------------------------------//
        public void addNotify()         {
                // Record the size of the window prior to calling parents addNotify.
                Dimension d = getSize();
                super.addNotify();
                if (fComponentsAdjusted)      return;
                setSize(getInsets().left + getInsets().right + d.width, getInsets().top +
getInsets().bottom + d.height);
                Component components[] = getComponents();
                for (int i = 0; i < components.length; i++){
                        Point p = components[i].getLocation();
                        p.translate(getInsets().left, getInsets().top);
                        components[i].setLocation(p);}
```

```
                fComponentsAdjusted = true;}
//-------------------------------------------------------------------------//
        // Used for addNotify check.
        boolean fComponentsAdjusted = false;
        java.awt.Button StartButton = new java.awt.Button();
        java.awt.Button StopButton = new java.awt.Button();
        java.awt.Choice FailureSelection = new java.awt.Choice();
        java.awt.Choice CategorySelection = new java.awt.Choice();
        java.awt.Label SelectCategory = new java.awt.Label();
        java.awt.Label SelectFailure = new java.awt.Label();
        java.awt.Label Status = new java.awt.Label();
        java.awt.TextArea StatusTextArea = new java.awt.TextArea("",178,339,0);
        java.awt.Label LabelArguments = new java.awt.Label();
        java.awt.TextField ArgumentsTextField = new java.awt.TextField();
        java.awt.Button ResetButton = new java.awt.Button();
        JScrollPane jscrollpane1 = new JScrollPane(StatusTextArea);
        java.awt.MenuBar mainMenuBar = new java.awt.MenuBar();
        java.awt.Menu menu1 = new java.awt.Menu();
        java.awt.MenuItem aboutMenuItem = new java.awt.MenuItem();
        java.awt.MenuItem menuItemSetupEnvironment = new java.awt.MenuItem();
        java.awt.MenuItem exitMenuItem = new java.awt.MenuItem();

//*************************************************************************//
        class SymWindow extends java.awt.event.WindowAdapter {
                public void windowClosing(java.awt.event.WindowEvent event){
                        Object object = event.getSource();
                        if (object == MainFailuresGUI.this)
                                MainFailuresGUI_WindowClosing(event);} }
//*************************************************************************//
void MainFailuresGUI_WindowClosing(java.awt.event.WindowEvent event)
        {try {      this.dispose();   } catch (Exception e) {}     }

//*************************************************************************//
        class SymAction implements java.awt.event.ActionListener{
                public void actionPerformed(java.awt.event.ActionEvent event){
        Object object = event.getSource();
                        if (object == aboutMenuItem)
                                aboutMenuItem_ActionPerformed(event);
                        else if (object == exitMenuItem)
                                exitMenuItem_ActionPerformed(event);
                        else if (object == ResetButton)
                                ResetButton_ActionPerformed(event);
                        else if (object == menuItemSetupEnvironment)
                                menuItemSetupEnvironment_ActionPerformed(event);
                        else if (object == StartButton)
                                StartButton_ActionPerformed(event);
                    else if (object == StopButton)
                                StopButton_ActionPerformed(event);}}

void aboutMenuItem_ActionPerformed(java.awt.event.ActionEvent event){
        try {(new AboutDialog(this, true)).setVisible(true);
```

```
                } catch (Exception e) {}}
//---------------------------------------------------------------------------//
        void exitMenuItem_ActionPerformed(java.awt.event.ActionEvent event){{
        try {    this.dispose();} catch (Exception e) {}}
//---------------------------------------------------------------------------//
        void ResetButton_ActionPerformed(java.awt.event.ActionEvent event){
                try {
                        PerformedSelection = false;
                        CategorySelection.setEnabled(true);
                        FailureSelection.setEnabled(true);
                        ArgumentsTextField.setText("");
                        StartButton.setEnabled(true);
                        StopButton.setEnabled(true);
                        StatusTextArea.setText("");
                        StatusTextArea.setCaretPosition(0);
                              setStatusAreaText("Reset is selected");
                              FailureSelection.removeAll();
                              FailureSelection.add("Displays the simulated failures");
                              try {      CategorySelection.select(0);                    }
                      catch (IllegalArgumentException e) { }
                } catch (java.lang.Exception e) {}     }
//---------------------------------------------------------------------------//
void StartButton_ActionPerformed(java.awt.event.ActionEvent event){
    PerformedSelection = true;
    SelectionType = "Simulation";
    setStatusAreaText("Start simulation is selected");
    WorkerExtentionAgentReference.setParameters(getSelectionParameters());     }
//---------------------------------------------------------------------------//
void StopButton_ActionPerformed(java.awt.event.ActionEvent event)  {
   PerformedSelection = false;
   setStatusAreaText("Stop simulation is selected\n");
   setStatusAreaText("All active moniotors will be terminated");
                        StatusTextArea.setText("");
                        StatusTextArea.setCaretPosition(0);
                              setStatusAreaText("Clear is selected");
  // WorkerExtentionAgentReference.TerminateMonitors();    }

//*************************************************************//
class SymItem implements java.awt.event.ItemListener{
                public void itemStateChanged(java.awt.event.ItemEvent event){
                        Object object = event.getSource();
                        if (object == CategorySelection)
                                CategorySelection_ItemStateChanged(event);
                        else if (object == FailureSelection)
                                FailureSelection_ItemStateChanged(event);}}
void CategorySelection_ItemStateChanged(java.awt.event.ItemEvent event){
                CategorySelection_ItemStateChanged_Interaction1(event);
                CategorySelection_ItemStateChanged_Interaction2(event); }
//---------------------------------------------------------------------------//
void CategorySelection_ItemStateChanged_Interaction1(java.awt.event.ItemEvent event){
                try {
```

```
        CategorySelection.setEnabled(false); } catch (java.lang.Exception e) {}}
//-------------------------------------------------------------------------//
void CategorySelection_ItemStateChanged_Interaction2(java.awt.event.ItemEvent event){
        try {           int SelectedIndex = CategorySelection.getSelectedIndex();
                        String SelectedItem = CategorySelection.getSelectedItem();
                        switch (SelectedIndex){
                        case 0 :
                        FailureSelection.removeAll();
                        FailureSelection.add("Correspondent Unavailable");
                    FailureSelection.add("Remote Communication Failure");
                    break;
                        case 1 :
                        FailureSelection.removeAll();
                        FailureSelection.add("Resource Unavailable");
                        ArgumentsTextField.setText("");
                        setStatusAreaText("Enter resource :");
        break;
        case 2 :
        FailureSelection.removeAll();
        FailureSelection.add("Collaborator Unavailable");
        break;
        case 3 :
        FailureSelection.removeAll();
        FailureSelection.add("Logic Unavailable");
                    FailureSelection.add("Logic Update Required");
        break;
        case 4 :
        FailureSelection.removeAll();
                    FailureSelection.add("Migration Denied");
                    FailureSelection.add("RunTime Environment Unavailable");
                    FailureSelection.add("Network Node Inaccessible");
                        ArgumentsTextField.setText("");
                  setStatusAreaText("\nEnter network location,alternative
node[<syntax>socket://nodeAddress:portNumber/],Reaattempt period");

        break;
        case 5 :
        FailureSelection.removeAll();
        FailureSelection.add("Critical Resource Utilization");
        break;
        case 6 :
        FailureSelection.removeAll();
                    FailureSelection.add("Migration Denied");
                    FailureSelection.add("RunTime Environment Unavailable");
                    FailureSelection.add("Network Node Inaccessible");
                    FailureSelection.add("Collaborator Unavailable");
                    FailureSelection.add("Correspondent Unavailable");
                    FailureSelection.add("Remote Communication Failure");
                    FailureSelection.add("Resource Unavailable");
                    FailureSelection.add("Logic Unavailable");
                    FailureSelection.add("Logic Update Required");
```

```
                    FailureSelection.add("Critical Resource Utilization");
                    break;          }
                } catch (java.lang.Exception e) {                   }         }
//-------------------------------------------------------------------------//

void menuItemSetupEnvironment_ActionPerformed(java.awt.event.ActionEvent event)   {
            (new InteractionFrame()).setVisible(true); }

public void setSelectionStatus(boolean SelectionStatus){
 PerformedSelection= SelectionStatus; }

public  String [] getSelectionParameters(){
   String []  returnParameters = new String [3];
   returnParameters[0]= SelectionType;
      if (!FailureSelection.getSelectedItem().equalsIgnoreCase("Displays the simulated
failures"))       {
      returnParameters[1]= FailureSelection.getSelectedItem();       }
      else returnParameters[1]= "null";
      if (ArgumentsTextField.getText().equalsIgnoreCase(""))       {
         returnParameters[2]=null;       }
      else returnParameters[2]= ArgumentsTextField.getText();
  return returnParameters;    }
//-------------------------------------------------------------------------//
 public void setStatusAreaText (String TextToDisplay){
  StatusTextArea.append(TextToDisplay+"\n"); }
   boolean PerformedSelection = false;
   String  SelectionType;
   WorkerExtentionAgent WorkerExtentionAgentReference;
}
```

34. ReadWrite

```
package uk.ac.surrey.callisto.pm.adaptation.UpdateProperties;
import java.io.*;
import java.io.Writer.*;
import java.util.*;
public class ReadWrite extends Object {
//-------------------------------------------------------------------------//
public boolean write (String Server1IP,String Server2IP,String ClassPath , boolean
UpdateInitialization, boolean UpdateClassPath){
      boolean ReturnBoolean=true;
      String ClassPathAndFileName = "./lib/ClassPath.properties";
      String ClassPathAndFileNameParticipatingServers =
"./lib/ParticipatingServers.properties";
      try {        FileWriter file_to_write = new
FileWriter(ClassPathAndFileNameParticipatingServers,UpdateInitialization);
          if (!Server1IP.equalsIgnoreCase(""))file_to_write.write(Server1IP+"|");
          if (!Server2IP.equalsIgnoreCase(""))file_to_write.write(Server2IP+"|");
          if (!ClassPath.equalsIgnoreCase(""))
            if (UpdateClassPath)
          file_to_write.close();
```

```
      } catch (Exception writeErr){
         writeErr.printStackTrace();
         ReturnBoolean=false;          }
    if  (UpdateClassPath)
            try {FileWriter FileClassPath = new FileWriter(ClassPathAndFileName, false);
                  FileClassPath.write(ClassPath+"|");
                  FileClassPath.close();
               } catch (Exception writeErr){
                  writeErr.printStackTrace();
                  ReturnBoolean=false;                }
         return ReturnBoolean;    }
//-------------------------------------------------------------------------//
 public Vector read (String path){
      Vector v = new Vector(40);
      try{  BufferedReader in = new BufferedReader(new FileReader(path));
      String s_read = in.readLine();
      StringTokenizer delimitered = new StringTokenizer(s_read,"|");
      while (delimitered.hasMoreTokens())  v.addElement(delimitered.nextToken());
      in.close();      }catch(Exception error){        }
      v.trimToSize();
    return v;     }
//-------------------------------------------------------------------------//
public boolean Update (String ClassPathAndFileName,String StringToUpdate, String
StringValue){
    return true;     }   }
```

35. InteractionFrame

```
package uk.ac.surrey.callisto.pm.adaptation.UpdateProperties;
import java.awt.*;
import java.util.*;
public class InteractionFrame extends Frame{
public  static String [] ParticipatingServersRead ;
public InteractionFrame(){
                  ParticipatingServersRead = readIPAddressesOfAvailableLocation();
                  setLayout(null);
                  setBackground(java.awt.Color.lightGray);
                  setSize(546,452);setVisible(false);add(checkboxClasspath);
                  checkboxClasspath.setBounds(108,192,24,28);
                  add(checkboxServers);
                  checkboxServers.setBounds(108,120,24,28);
                  checkboxServers.setState(true);
                  ComboUpdatePath.setText("UpdatePath");
                  add(ComboUpdatePath);
                  ComboUpdatePath.setBounds(22,193,74,28);
                  UpdateInitializationTrueFalse.setText("New entries ?");
                  add(UpdateInitializationTrueFalse);
                  UpdateInitializationTrueFalse.setBounds(22,120,87,28);
                  CommitChangeButton.setLabel("Commit Changes");
                  add(CommitChangeButton);
```

```
CommitChangeButton.setBackground(java.awt.Color.lightGray);
CommitChangeButton.setBounds(143,392,112,39);
DoneButton.setLabel(" Exit ");
add(DoneButton);
DoneButton.setBackground(java.awt.Color.lightGray);
DoneButton.setBounds(290,391,112,39);
add(textFieldServer1);
textFieldServer1.setFont(new Font("SansSerif", Font.PLAIN, 12));
textFieldServer1.setBounds(144,50,378,28);
add(textFieldServer2);
textFieldServer2.setFont(new Font("SansSerif", Font.PLAIN, 12));
textFieldServer2.setBounds(144,85,378,28);
label1IPAddress.setText("IP Addresses of participating nodes");
add(label1IPAddress);
label1IPAddress.setFont(new Font("SansSerif", Font.PLAIN, 12));
label1IPAddress.setBounds(144,24,228,24);
label1ServerIPAddress.setText("Home Node");
add(label1ServerIPAddress);
label1ServerIPAddress.setBounds(22,48,86,28);
label2ServerIPAddress.setText("Destination Node");
add(label2ServerIPAddress);
label2ServerIPAddress.setBounds(22,84,110,28);
labelpathToClasses.setText("Path to project classes");
add(labelpathToClasses);
labelpathToClasses.setBounds(144,228,144,28);
frameButton.setEnabled(false);
add(frameButton);
frameButton.setBackground(java.awt.Color.lightGray);
frameButton.setBounds(12,12,525,144);
textFieldClassPath.setText("C:\\java\\bin\\");
textFieldClassPath.setEnabled(false);
add(textFieldClassPath);
textFieldClassPath.setFont(new Font("SansSerif", Font.PLAIN, 12));
textFieldClassPath.setBounds(144,264,380,28);
label2Classpath.setText("Classpath");
add(label2Classpath);
label2Classpath.setBounds(22,264,98,28);
frameButton2.setEnabled(false);
add(frameButton2);
frameButton2.setBackground(java.awt.Color.lightGray);
frameButton2.setBounds(12,180,525,180);
setTitle("System Properties Update");
textFieldServer1.setText(ParticipatingServersRead[0]);
textFieldServer2.setText(ParticipatingServersRead[1]);
SymWindow aSymWindow = new SymWindow();
this.addWindowListener(aSymWindow);
SymAction lSymAction = new SymAction();
DoneButton.addActionListener(lSymAction);
CommitChangeButton.addActionListener(lSymAction);
textFieldServer1.addActionListener(lSymAction);
textFieldServer2.addActionListener(lSymAction);
```

```
                SymItem lSymItem = new SymItem();
                checkboxClasspath.addItemListener(lSymItem);      }
public InteractionFrame(String title)  {
                this();
                setTitle(title);  }
        static public void main(String args[]){
                ParticipatingServersRead = readIPAddressesOfAvailableLocation();
                try{(new InteractionFrame()).setVisible(true);         }
                catch (Throwable t){
                        System.err.println(t);
                        t.printStackTrace();
                        System.exit(1);}}
        boolean fComponentsAdjusted = false;
        java.awt.Checkbox checkboxClasspath = new java.awt.Checkbox();
        java.awt.Checkbox checkboxServers = new java.awt.Checkbox();
        java.awt.Label ComboUpdatePath = new java.awt.Label();
        java.awt.Label UpdateInitializationTrueFalse = new java.awt.Label();
        java.awt.Button CommitChangeButton = new java.awt.Button();
        java.awt.Button DoneButton = new java.awt.Button();
        java.awt.TextField textFieldServer1 = new java.awt.TextField();
        java.awt.TextField textFieldServer2 = new java.awt.TextField();
        java.awt.Label label1IPAddress = new java.awt.Label();
        java.awt.Label label1ServerIPAddress = new java.awt.Label();
        java.awt.Label label2ServerIPAddress = new java.awt.Label();
        java.awt.Label labelpathToClasses = new java.awt.Label();
        java.awt.Button frameButton = new java.awt.Button();
        java.awt.TextField textFieldClassPath = new java.awt.TextField();
        java.awt.Label label2Classpath = new java.awt.Label();
        java.awt.Button frameButton2 = new java.awt.Button();
class SymWindow extends java.awt.event.WindowAdapter {
    public void windowClosing(java.awt.event.WindowEvent event){
                        Object object = event.getSource();
                        if (object == InteractionFrame.this)
                                InteractionFrame_WindowClosing(event);}  }
void InteractionFrame_WindowClosing(java.awt.event.WindowEvent event){
        try {    (new QuitDialog(this, true)).setVisible(true);
                } catch (Exception e) {}}
class SymAction implements java.awt.event.ActionListener{
                public void actionPerformed(java.awt.event.ActionEvent event){
                        Object object = event.getSource();
                        if (object == DoneButton) DoneButton_ActionPerformed(event);
                        else if (object == CommitChangeButton)
                                CommitChangeButton_ActionPerformed(event);}}
void DoneButton_ActionPerformed(java.awt.event.ActionEvent event)     {
                try {    (new QuitDialog(this, true)).setVisible(true);
                } catch (Exception e) {}}
void CommitChangeButton_ActionPerformed(java.awt.event.ActionEvent event) {
      if (ServerStateChanged==true){
                ReadWrite WriteToFile = new ReadWrite();
```

```
                boolean returnType =
WriteToFile.write(textFieldServer1.getText(),textFieldServer2.getText(),textFieldClassPath.g
etText(),checkboxServers.getState(),checkboxClasspath.getState());
                ServerStateChanged=false;    }}
        public String getProjectPath(){    return textFieldClassPath.getText();}
class SymItem implements java.awt.event.ItemListener{
                public void itemStateChanged(java.awt.event.ItemEvent event){
                        Object object = event.getSource();
                        if (object == checkboxClasspath)
                                checkboxClasspath_ItemStateChanged(event);
                        if (object == textFieldServer1)
                         textFieldUpdatedActionPerformed(event);
                        if (object == textFieldServer2)
                         textFieldUpdatedActionPerformed(event);} }
void textFieldUpdatedActionPerformed(java.awt.event.ItemEvent event){
ServerStateChanged= true;}
void checkboxClasspath_ItemStateChanged(java.awt.event.ItemEvent event){
                try {
                        textFieldClassPath.setEnabled(checkboxClasspath.getState());
                        UpdateFlag = checkboxClasspath.getState();
                } catch (java.lang.Exception e) {}}
public static String [] readIPAddressesOfAvailableLocation (){
    ReadWrite ReadFromFile = new ReadWrite();
   Vector  Servers = ReadFromFile.read("./lib/ParticipatingServers.properties");
   String [] ParticipatingServers= new String [Servers.size()];
                try   {
        for( int index=0; index <(Servers.size());index++){
ParticipatingServers[index]="socket://"+(String)Servers.elementAt(index)+":7000/destination
";  System.out.println("Participating servers are :  "+ ParticipatingServers[index]);       }       }
      catch (Exception writeErr){ writeErr.printStackTrace();   }
return ParticipatingServers; }
boolean UpdateFlag = false;
boolean ServerStateChanged=false;
boolean flagToClassPathUpdate = false;}
```