THE PROBLEMATIC OF DISTRIBUTED SYSTEMS SUPERVISION - AN EXAMPLE : GENESYS

Jean-Eric Bohdanowicz[1], Stefan Wesner[2], Laszlo Kovacs[3], Hendrik Heimer[4], Andrey Sadovykh[5]
[1]*EADS SPACE Transportation,* [2]*HLRS- Stuttgart University,* [3]*MTA SZTAKI,* [4]*NAVUS GmbH,* [5]*LIP6*

Abstract: This chapter presents the problematic of the distributed systems supervision through a comprehensive state-of-the-art. Issues are illustrated with a case study about an innovative and generic supervision tool, GeneSyS.

Key words: Supervision, distributed management, state-of-the-art, GeneSyS, intelligent agent, Web-Services.

1. INTRODUCTION

«Today, the performance of information systems directly governs company competitiveness », such is the report that can be pulled from the evolution of the information technologies.

The supervision of the computer infrastructure becomes therefore an element of vital importance for the whole set of companies. In a context where «business» is tangled closely with the information system to give birth to «e-business», it has been necessary to erase the border between technology and business in order to provide some indicators valuable for the company managers.

As a result, near the end of 1990's, the concept of «frameworks» has been considered like the solution for the supervision of distributed systems and applications.

These frameworks can be compared with ERP's although their respective domains of application are different : their ambition is to manage the whole information system through an integrated unique offer.

Since several months, many study groups like Gartner Group or Meta Groups grant to make a mitigated balance on the usage of these frameworks. The complexity of implementation of these platforms seems to have made indeed fail close to 3 projects on 4. Besides, the very elevated costs of licenses and deployment have limited such a solution perimeter to big companies.

Historically, the principles of network supervision are older than those governing the frameworks and mainly based upon the SNMP protocol (and its extensions). Numerous network monitoring offers are today available on the market. That's why the editors leaders of the sector, conscious of the competitive hazard that the frameworks represent, made their offer to evolve toward system and applications monitoring. However, network supervision platforms do not constitute the ideal basis for systems and application supervision.

Therefore, most users are today facing a spiny problem : there is no available pragmatic approach for the global network, systems and applications supervision and, with the international dimension of today's projects, this supervision of distributed systems is becoming necessary and primordial and it requires a wide panel of services :

- **Application management** including deployment, set-up, start, stop, hold/resume and configuration management (for instance, for redundancy management purpose),
- **Time synchronisation,**
- **Network management** including parameterisation and performances (e.g. dynamic control of bandwidth allocation) **and monitoring,**
- **Security,**
- **Archiving.**

These services requirements lead to have a set of independent software components designed in a distributed way using the following technologies :

- the *applications,* distributed on the remote sites,
- the *groupware,* allowing the collaborative working,
- the *middleware,* managing the distribution (like CORBA, HLA…),
- the *synchronisation,* giving the same time reference to all the sites such as NTP,

- the *security,* protecting the data transmission and access control to the resources,
- the *network management,* using mainly SNMP,
- the *network layer,* interconnecting the remote sites and the various equipment within a given site,
- the *hardware platforms,* implementing the distributed system services.

What is needed is a convergence of the different supervision offers. This has been observed toward a "single vision" consisting in an enterprise type approach. This new tendency is related to the will of, on one hand, the "systems supervision" solutions editors to open their products to the network and, on the other hand, the "network supervision" developers to integrate in their solutions a system monitoring.

But today, none of these available solutions was specifically designed to fulfil its principal task: the global supervision with an "enterprise" point of view. Historically, these solutions are mainly proprietary, most of the time made of a pool of offers acquired by the mean of external growth and aimed at covering a large functional scale.

Today, the supervision of distributed systems is mainly done on a case by case basis and also mainly at independent technical services levels. The distributed applications used on these systems are supervised in a very limited way. Furthermore, the user-interface is often questionable as it requires expert people and suffers from a lack of automation and user friendliness.

The main purpose of this chapter is to present:

- in a first part, a state-of-the-art of the existing technologies, features, standards, protocols, tools involved in the distributed systems supervision world,
- in a second part, a new, innovative, open and generic approach which is detailed through the description of the GeneSyS project.

2. STATE OF THE ART

The state of the art section intends to give an overview of some of the main standards, protocols and tools used in the distributed systems supervision field. This section is also providing a list of current supervision frameworks and related research projects.

2.1 Standards in Distributed Management

This paragraph describes two of the most known standards: SNMP and JMX. In addition, DMTF general use standards are listed at the end of this section.

2.1.1 SMTP

The network management technologies have been developed during the whole history of networks. The most known of them are High-level Entity Management System (HEMS), Simple Gateway Monitoring Protocol (SGMP) and Common Management Information Protocol (CMIP). These technologies highly contributed to the Simple Network Management Protocol (SNMP).

SNMP is the most widely used protocol for the management of IP based networks. Its concept also allows management of end systems and applications using specific Agents and Management Information Bases (MIB). Although SNMP version 3, covering security issues, was already released, the version 1, due to its robustness, is still widely used.

SNMP is an application level protocol on top of UDP. SNMP managed network consists of three major components (Figure 1): managed devices, agents and Network Management Systems (NMS). The managed devices can be hosts, network interfaces, routers, bridges, hubs and etc. The agents are the program components running in the managed devices. Agents collect an information about managed devices and make it available for NMS by the mean of SNMP. The NMS executes the management applications to monitor and control the managed devices.

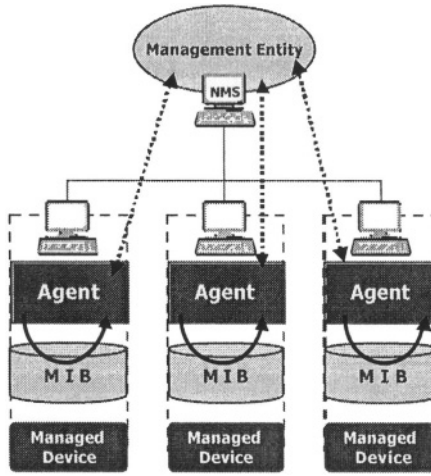

Figure 1. SNMP Managed Network

The management capability of the devices can be quite poor due to, for example, slow CPU or memory limitations. That is why the agent should minimise its impact on the managed device. Moreover, all calculation and monitoring data processing is centralised in Network Management System which, in addition, implements graphical user interface (GUI).

Communication between Agents and NMS is assured by the Network Management Framework protocol. This protocol supports the Query/Response mechanism when Agents send parameters values upon request of the NMS, as well as the Subscribe mechanism, which deals with asynchronous messages sent by Agent to NMS when a particular event happens.

The Managed Devices are monitored and controlled using four basic SNMP commands: read, write, trap, and traversal operations.

- The read command is used by an NMS to monitor managed devices. The NMS examines different variables that are maintained by managed devices.
- The write command is used by an NMS to control managed devices. The NMS changes the values of variables stored within managed devices.
- The trap command is used by managed devices to asynchronously report events to the NMS. When certain types of events occur, a managed device sends a trap to the NMS.
- Traversal operations are used by the NMS to determine which variables a managed device supports and to sequentially gather information in variable tables, such as a routing table.

SNMP lacks any authentication capabilities, which results in vulnerability to a variety of security threats. These include masquerading occurrences, modification of information, message sequence and timing modifications, and disclosure. Masquerading consists of an unauthorised entity attempting to perform management operations by assuming the identity of an authorised management entity. Modification of information involves an unauthorised entity attempting to alter a message generated by an authorised entity so that the message results in unauthorised accounting management or configuration management operations. Message sequence and timing modifications occur when an unauthorised entity reorders, delays, or copies and later replays a message generated by an authorised entity. Disclosure results when an unauthorised entity extracts values stored in managed objects, or learns of noticeable events by monitoring exchanges between managers and agents. Because SNMP does not implement authentication, many vendors do not implement Set operations, thereby reducing SNMP to a monitoring facility.

2.1.2 JMX

The Java Management eXtensions (JMX) is a SUN specification describing the design patterns of smart Java agents for application and network management. The specification includes the architecture, the design patterns, APIs and core services. The JMX provides Java developers with means to instrument Java code and create smart Java agents and management applications. The JMX components also provide means for extension of existing Java based management middleware. It is already planned to integrate JMX into such systems as:

- WBEM (JSR-000048 WBEM Services Specification for CIM/WBEM manager and provider APIs)[1]
- SNMP Manager API (currently reviewed by the Java Community Process)

The JMX propose a three layers architecture comprising:

- Instrumental level (interfaces to manageable resources),
- Agent level (Server),
- Distributed Services level (External Applications).

The following figure is clarifying the relations between these levels and their components.

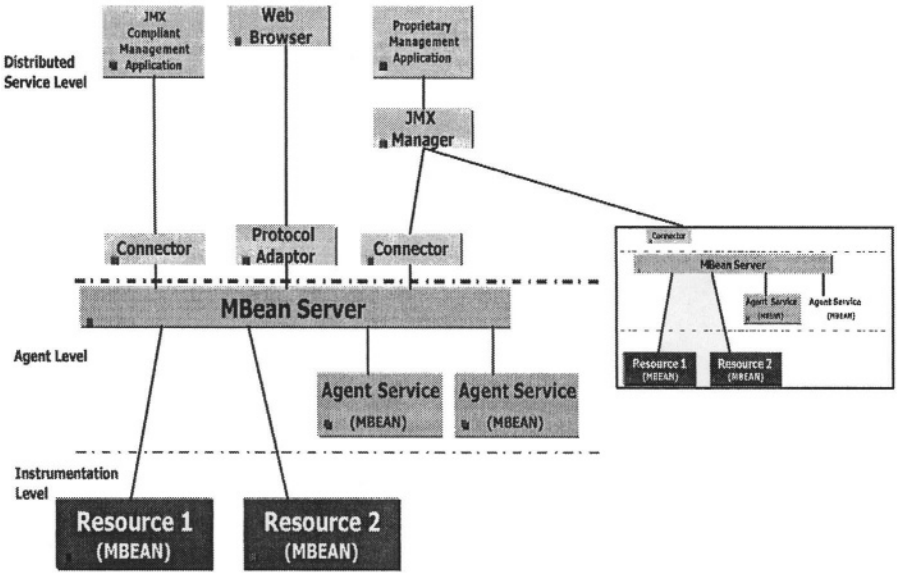

Figure 2. Relationship between components of the JMX architecture

Instrumentation Level: This level deals with components to be managed. A JMX manageable component can be an application, a service, a device, a user and etc. An instrumentation can be done through a Java interface or thin Java wrapper by means of implementation of Manageable Beans (MBeans). An MBean is a special Java Bean that should be implemented with stricter design pattern than a common Java Bean. The main aim of the instrumentation is to provide services to the agent level (to a Mbean Server). This server manage all communications between the MBeans.

Moreover, the instrumentation level supports publish/subscribe communication model (notification mechanism) which is a standard for Java Beans, this mechanism is used to propagate the notifications events to the upper levels.

JMX is a quite portable system, since it requires that resource is compatible only with JDK 1.1.x, EmbeddedJava, PersonalJava or Java2. It means that wide rage of resources can be managed. Besides, JMX ensures high level of automation of management for such instrumented resources.

Agent Level: This level deals with the management agents. The Agents can directly access the instrumented resources to control them and to publish them to Management applications on upper level.

The JMX agent consists of an MBean Server and a set of services for handling MBeans. Due to this separation, the agent and instrumented resources can be placed on different hosts. Similar to the approach at the instrumentation level, the JMX agent is designed to be independent regarding the Management application that is using the agent.

Distributed Services Level: The blue blocs on Figure 2 represent the Distributed Services level which deals with Management applications. However, this level is not yet well defined in the JMX specification. This level defines the interfaces needed for the implementation of JMX managers that are purposed to integrate managed resources seamless to their environment. In addition, the components named Connector and Protocol Adapter are used to provide information to different clients.

2.1.3 Distributed Management Task Forces Standards

The Distributed Management Task Forces (DMTF) released several specifications widely used in modern supervision frameworks, like OpenView and Unicenter. The most relevant are:

Common Information Model <http://www.dmtf.org/standards/cim> (CIM) - a common data model of an implementation-neutral schema for describing overall management information in a network/enterprise environment.

Web-Based Enterprise Management <http://www.dmtf.org/standards/wbem> (WBEM) - a set of management and Internet standard technologies developed to unify the management of enterprise computing environments.

2.2 Supervision Frameworks

This section gives an outline of the current frameworks in the supervision domain focusing on the 3 main frameworks : Tivoli, Unicenter and HP Openview. A short overview of other existing frameworks is also provided.

2.2.1 Tivoli (IBM) (http://www.tivoli.com)

The Tivoli framework is built on a CORBA compliant middleware, it is more dedicated to distributed system administration : software distribution, remote configuration, remote control, remote monitoring. Its design is proprietary, thus it does not support standard protocols (like SNMP) but it

can be interfaced with the IBM's network analysis product Netview in order to enhance the field of operations.

2.2.1.1 Overview

The ***Tivoli Management Environment*** (**TME**) is a product line whose base component is the Tivoli Management Environment ***Framework.*** Using the Tivoli Framework and a combination of TME applications, it is possible to manage large distributed networks with multiple operating systems, various network services, diverse system tasks and constantly changing nodes and users.

The TME Framework provides a set of common services or features that are used by the TME applications installed on the Framework. Examples of services provided by the Framework are:

- The DHCP service.
- The Task library through which tasks can be created and executed on multiple TME resources.
- A scheduler that makes it possible to schedule all TME operations including the execution of tasks created in the TME Task library.
- The RDBMS interface module (RIM) that enables some TME applications to write application specific information to relational databases.
- The query facility that allows search and retrieval of information from a relational database.

TME applications installed on the TME Framework are enabled to use the services provided by the Framework.

TME provides centralised control of a distributed environment, which can include mainframes, UNIX or NT workstations, and PCs. A single system administrator can perform the following task for bunches of networked systems:

- Manage user and group accounts
- Deploy new or upgrade existing software
- Inventory existing system configuration
- Monitor the resources of systems either inside or outside the TME environment
- Manage internet and intranet access and control
- Manage third-party applications.

2.2.1.2 TME Management Services

The TME Framework enables installation and creation of several management services such as:

- **TMR Server –** It includes the libraries, binaries, data files, and graphical user interface needed to install and manage a TME environment. TMR servers maintain the TMR server database and co-ordinate all communications with the TME managed nodes.
- **Managed Node –** A TME Managed Node runs the same software that runs on a TMR Server. Managed nodes maintain their own databases, which can be accessed by the TMR server. When managed nodes communicate directly with other managed nodes, they perform the same communication or security operations performed by the TMR Server. The primary difference between a TMR server and a managed node is the size of the database.
- **Endpoint gateway** – An endpoint gateway controls all communications with and operations on TME endpoints. A single gateway can support communications with thousands of endpoints. A gateway can launch method on an endpoint or run methods on the endpoint's behalf. Created on an existing managed node, the gateway is a proxy managed node that provides access to the endpoint methods and provides the communications with the TMR server that the endpoint occasionally require.
- **Endpoint –** An endpoint is any system that runs an endpoint service (daemon). Typically, an endpoint is installed on a machine that is not used for daily management operations. Endpoints run a very small amount of software and do not maintain a database. The majority of systems in most TME installations are endpoints.

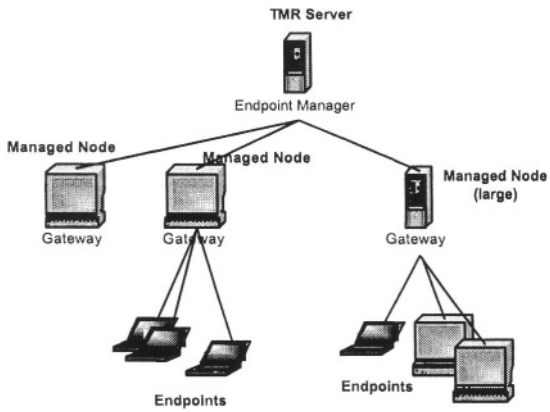

Figure3. TME Framework Nodes

Every TME framework installation begins with a TMR Server, which is just a special case of a managed node with some additional responsibilities, such as locating objects within the TME distributed database and performing authentication for method invocations. For every method invocation, the TMR server must be contacted to locate the object and authenticate the method invocation. In addition, the TMR server is the point at which much of the inter-TMR communication takes place.

2.2.1.3 Communications and networks

TME provides a distributed environment on top of which system management application run. This environment consists of one or more machines that perform operations in a distributed and parallel fashion. Each machine in a TMR has a long-running service, or daemon, called the **oserv** that communicates with other TME services, or daemons, on other machines in a peer-to-peer based manner. An operation initiated on one machine may start multiple operations on machines across the network, all running in parallel to complete their portion of the overall task.

The configuration of TMRs and the location of file servers have a significant impact on the performance of the TME installation. For example, if two sites are connected through a slow line over which TME requests and operations are run, each site should then be a TMR and have a local file server with the appropriate TME binaries. In this manner, the only traffic that passes over the slow line between the sites are management requests, not large amounts of data or requests for information from a remote TME server.

Due to the distributed architecture, it is important that the communications and network function efficiently. The TME server speeds up error and timeout scenarios as well as ensures reliable and accurate error handling and recovery (e.g. it can track machines that are temporarily unavailable due to network problems).

TME provides a service called **Multiplexed Distribution** (Mdist) service, to enable synchronous distributions of large amounts of data to multiple targets in an enterprise. The Mdist service is used by a number of TME applications, such as TME Software Distribution, to maximise data throughput across large, complex networks.

During a distribution of data to multiple targets, Mdist sets up a distribution tree of communication channels from the source host to targets through repeaters. Mdist limits its own use of the network, as configured through repeater parameters, to help prevent intense network activity that can stress network bandwidth for periods of time.

There are fundamentally two types of network communication services available in TME – all other communications that use the TME Framework are built on top of these two communications services:

- Inter-Object Messaging (IOM)
- Inter-dispatcher communication (objcall service)

IOM represents the direct communication between object implementations. Once two object implementations are running, they can establish an IOM channel between them for the purpose of bulk data transfers. This channel is preferred from bulk data transfers, since sending large amounts of data as arguments to methods (via dispatcher) is slow and inefficient. An IOM channel usually only lasts as long as it takes to transfer the data it was created to accommodate.

Examples of IOM usage are: software, profiles and tasks distribution, file transfers between managed node files, TME database backups, TME desktop (GUI) communications.

As the primary type of communication in TME, the objcall service is used by all method invocations. When two dispatchers communicate, inter-dispatcher connections are sustained: the connection isn't broken unless the network breaks it, or unless one of the dispatchers is restarted.

An example of inter-dispatcher communication is illustrated in the following figure, which shows communications between two dispatchers and two object implementations.

2.2.2 Unicenter (Computer Associates) (http://www.cai.com)

The Unicenter framework is based on a central object repository containing all devices managed by the platform. Its implementation is more open than Tivoli's, it admits use of various protocols and let developers adding some extension modules.

2.2.2.1 Unicenter architecture

Unicenter architecture consists of the following:

- ***Real Work Interface*** – graphical user interface driven by a "Common Object Repository".

  Unicenter TNG's Real World Interface allows management applications to identify the business resources they manage, as well as the relationships among those resources. It draws on the

Common Object Repository to generate management maps dynamically.

- ***Common Object Repository*** (CORE) – central storage mechanism for all components of Unicenter TNG, accessible by management functions and third-party applications.

  The CORE is the location where all Unicenter TNG management functions store information about managed resources, their properties and relationships. Third-party applications and all Unicenter TNG components access CORE. The CORE is an object-based repository, which is database independent and designed for multi-user and multi-system operations.

- ***Managers and agents*** – core management facilities that provide resource management throughout an enterprise and agents means to monitor and control all aspects of the business enterprise.

  To manage varieties of hardware and software, widely dispersed across a network and distributed across multiple disparate platforms, Unicenter TNG proposes an infrastructure comprised of agents and managers. Agents reside on or near the managed resources, gather data about the resources and filter the data to identify and report the most important information to managers.

  Managers may be located anywhere in the network. They analyse the information sent to them by agents, correlate the various pieces of information in the environment to discover trends and patterns and determine how to best control the managed resources in the context of management policies.

2.2.2.2 Unicenter TNG's distributed management approach

In Unicenter TNG's manager/agent architecture, the functions that use management information, control management actions and delegate management authority are architecturally separate from the functions that produce management data and act on behalf of managers. Many managers can monitor a single agent and vice versa. GUI can use the Common Object Repository and multiple managers can update that repository.

Manager's role - A manager is one of many software bosses in the enterprise management system. Managers issue requests to agents for data and then perform analyses and correlations on the data received about their management environment.

Unicenter TNG has for example the following managers: a workload manager, storage manager, asset manager, problem manager, software

distribution manager, configuration manager, file manager, calendar manager, report manager, user/security manager...

There is also a special manager called Distributed State Machine (DSM), which manage groups of agents that instrument resources. This manager is essential to integration of third-party agents.

Agent's role - Agents monitor information about one or more resources and relay that information to a manager under specific circumstances or criteria. Agents can periodically report to their managers or be asked (polled) for information by managers.

Unicenter TNG offers several agents right out of the box: DB2 agent, DCE agent, Informix agent, Ingres II agent, MVS agent, Netware agent, OpenEdition agent, OpenVMS agent, Oracle agent, OS/2 agent, OS/390 System agent, SQL Server agent, Sybase agent, Tandem NSK agent, Unix agent, Windows 3.1 agent, Windows 95 agent, Windows NT agent…

2.2.2.3 Unicenter TNG agent technology and integration

Unicenter TNG agent technology makes it possible to instrument practically any resource in an IT infrastructure. It provides facilities for creating custom agents. The open architecture supports agents created by other software vendors who have followed the Unicenter TNG agent specifications.

Unicenter TNG provides a SDK, which helps third parties to integrate their solutions into Unicenter TNG. The SDK consists of API organised as the following:

- WorldView (GUI) API
- Enterprise Management API
- Agent Factory

The ***Worldview API*** is comprised of the Real World Interface and the Common Object Repository. It provides utilities for customising the GUI without impacting the behaviour of the management applications.

The ***Enterprise Management API*** controls all the management functions and common services provided in Unicenter TNG and provides them for cross-application integration. It provides multi-platform management facilities for security, help desk, event management… Third-party management applications can share policies, request services from, and provide services to other management functions.

The ***Agent Factory API*** allows third parties to construct multi-platform, scalable manager/agent applications. These agents may also be deployed over the Internet and Intranets. It is a complete development environment for building agents that communicate with management applications using

SNMP. Within Unicenter TNG architecture, those management applications include WorldView and third-party applications at the Enterprise Management level.

2.2.2.4 Unicenter TNG's Agent Factory environment

The Agent Factory allows building a SNMP agent with minimum effort: only the code that is specific to the resources needs to be written. The functions that are common to any agent, such as encoding and decoding SNMP protocol data units and routing requests, are provided by a set of ***common services*** and a ***Distributed Services Bus.*** Agents in Unicenter TNG run within the Agent Factory environment, supported by the common service and the distributed Service Bus.

The Agent Factory provides the API libraries, executable code for the common services and Distributed Services Bus, utilities to configure and test agents. The common services consist of the executable code for three objects that perform the functions common to all agents: SNMP Gateway, SNMP Administrator, Object Store.

The ***Object Store*** consists of disk storage and the process that reads from and writes to that storage area. Object Store is designed to handle all incoming get and set requests by default. The API functions can be used to code a task that periodically calculates attribute values and send them to Object Store, where they are available whenever the SNMP Administrator receives a get or get-next request. Besides acting as a repository for current attribute values, Object Store also holds other critical agent data.

2.2.3 Openview (HP) (http://www.hp.com)

Since the begining dedicated to network supervision, the Openview environement has been augmented with many functionalities linked to systems and applications. Its implementation is based on SNMP and its design is closer to a software suite than a framework like IBM's or CA's.

2.2.3.1 Overview

HP Openview IT/Operations (ITO) is a software application that provides central operations and problem management for multi-vendor distributed systems.

ITO consists of a central management server in the form of a manager, which interacts with intelligent software-agents installed on the managed

systems (called nodes). Management status information, messages, and monitoring values are collected from such sources as system or application log files, SNMP traps, SNMP variables. Filters and thresholds are applied and the information is then converted into a standard format for presentation to the central management server. Once the information is retrieved, ITO can immediately initiate corrective actions and provide individual guidance for problem identification and further problem resolutions.

All management information and associated records needed for future analysis and audit are stored in a central repository called the History Database. It allows the automation of certain problem resolution processes.

2.2.3.2 ITO functioning

ITO monitors, controls and maintains systems in heterogeneous environments, by managing events, messages and actions. ITO uses events, messages, and actions to observe and control status, formulate and provide information, react to and correct problems.

When an event occurs, a message is generated as a result of that event. ITO performs event correlation on messages rather than on events: messages are copied rather than diverted to the correlation engine so that critical messages may avoid the possibility of being delayed or even lost in the correlation process. Messages are structured pieces of information, created by events. ITO intercepts and collects messages, and thereby is informed of events.

ITO message management can combine messages into logically related groups, bringing together messages from lots of related sources, providing status information about a class of managed objects or services. Other message management operations can classify and filter messages to ensure that important information is clearly displayed.

When an event occurs on a managed object, a message is created as a result. The ITO Agent on that managed node receives the message and filters it. It can then forward it and/or log it locally. If the message satisfies the filter, it is converted into ITO message format and forwarded to the management server. If a local action on the message is configured, it will be started. The management server can perform the following actions: assign the message to another message group, start non-local automatic actions configured for the message on the specified node, forward the message to external notification interfaces and trouble ticket service, escalate the message to another pre-configured management server. The active message is stored in the database and displayed in a Message Browser window in one

or more ITO display stations. When the message is acknowledged, it is removed from the active Browser and put in the history database.

2.2.3.3 ITO architecture

ITO architecture is comprised of the following:

- The management server.
- Managed nodes.

ITO software is divided into two basic components:

- Agents and sub-agents
- Managers.

The agent and sub-agent are located on the managed nodes and are responsible for generating messages, collecting and forwarding information, monitoring parameters.

The management software is located on the management server and communicates with, controls and directs the agents. It stores the central database and runs the graphical user interfaces.

The management server performs the central role of ITO. It collects data from managed nodes, managed and re-groups messages, calls the appropriate agent to start actions or initiate sessions on managed nodes, controls the history database for messages and performed actions, forwards messages, installs ITO agent software on managed nodes, intercepts SNMP traps.

2.2.3.4 Integration of applications into ITO

Existing applications can be integrated into ITO, at different levels, through various interfaces, to provide diverse capabilities and advantages:

- Application Desktop integration – applications are registered within ITO and represented by symbols in the application desktop window. Operators use these symbols daily to start applications and resolve problems.
- Event Integrations – applications can write messages to logfiles, use ITO API or send SNMP traps in order to manage events through ITO
- Action Integration – application start-ups can be incorporated into an automatic, or operator initiated action.
- Monitor Integration – monitoring applications such as scripts, programs, MIB variable based programs can be started by ITO and use API to return values. The monitored values can then be compared to threshold limits.

2.2.4 Other frameworks

Openmaster (Evidian-Bull) (http://www.evidian.com)

Openmaster was designed like a universal platform supporting a large field of protocols : SNMP, CMIS, CMIP … Network oriented, Evidian's Openmaster strategy seems today to focus on security purposes.

Nagios (OpenSource)

Nagios is an OpenSource framework that provides a flexible approach for centralised system monitoring. The core component is a Linux application that runs periodically a set of shell scripts. These scripts monitors different parameters (ping, system resource load, motherboard temperature, etc.) on local and remote hosts. A user-friendly GUI is secured by a web front-end.

2.3 Related Projects

This paragraph intends to present a list of related research project and organisations that are of interest when entering in the distributed systems supervision world.

Projects

ANDROID - The Active Distributed Open Infrastructure Development project provides a manageable programmable network infrastructure. The communication mechanism uses XML-based protocol. A genetic algorithm is used for intelligent policies based server management.

AgentScape - Scalable Resource Management for Multi-Agent Systems

MANTRIP - The Management Testing and Reconfiguration of IP based networks project is an example of Mobile Agent Technology (MAT) in the context of Network Management.

SHUFFLE - this project proposes an agent based approach to control resources in UMTS networks.

OPENDREAMS - The Open Distributed Reliable Environment architecture and Middleware for Supervision project was to satisfy the needs of advanced Supervision and Control Systems (SCSs) for the management of large equipment infrastructures such as telecommunication networks, electricity and water distribution networks, large buildings, etc. A Corba implementation was used as backbone assuring the interoperability and openness of the platform architecture.

WSDM - The OASIS Web Services Distributed Management Technical Committee defines web services management. This includes using

web services architecture and technology to manage distributed resources. The work is ongoing.

Agents management related projects:

AgentLight - Platform for Lightweight Agents project is dedicated to development of agent-based middleware for mobile devices using J2ME and FIPA compatible API.

AgentCities - This project is purposed to set up a world-wide network of always running FIPA test-bed platforms.

LEAP - The Lightweight Extensible Agent Platform is another project addressing the needs of mobile enterprises. The proposed architecture is based on JADE (Java FIPA implementation)

SAFIRA - The support Affective Interactions for Real-time Applications project provides a framework to enrich interactions and applications using a real-time multi-agent middleware.

Intelligent agents related projects:

Agent Academy - this project is concentrated on a data-mining framework for training intelligent agent. The project uses standards from FIPA and OMG, like FIPA ACL and KQML for agents' communication and OMG XMI and CWM MOF for data-mining.

PISA - The Privacy Incorporated Software Agent project deals with development of security software agents for the Internet and E-commerce.

RACING -The Rational Agent Coalitions for Intelligent Mediation of Information Retrieval on the Net is another example of agent-based data mining.

2.4 Intelligent Supervision

This section describes an innovative feature that needs to be implemented in the new supervision solution : the intelligent supervision.

Systems that leverage from simple monitoring system towards autonomous systems extend the basic components of data monitoring and potentially higher level components that analyse and interpret this basic data into higher layer information with elements that have the ability to react without further human intervention on a detected critical or failure situation. Such "intelligent" components need either pre-recorded knowledge or have to analyse a system.

We have identified three major elements of such a system:

- Case-Database solutions based on historical data and/or expert knowledge
- Topology Analysis for identifying root causes of failures
- System Behaviour Prediction

The following sections describe these elements in more details and provide an overview on the current state of the art in this area.

2.4.1 Case Database Approach

The Case Database approach for the management of distributed applications (as described in [2], [3]) is based on a set of cases containing specific symptom-cause pairs in a database. If a monitored situation fits or is similar to a symptom stored in the case database, the prepared solutions for such a case can be executed. The major problem to be solved in this kind of systems is finding appropriate metrics for evaluating the level of similarity between the monitored symptoms and the one stored in the database. Another critical point is the size and quality of the case database. Very often, historical data for example from a trouble ticket system are used to feed such databases (see[4]). Another approach correlates basic events to higher-level events that allow reacting either a system administrator or potentially an intelligent component correctly. This is either realised using programming language constructs (see [5]) or the system to be monitored is modelled with its event behaviour (cf. [6]).

2.4.2 Topology Analysis

Distributed applications consist out of interdependent components on different levels ranging from network, middleware, up to the application layer. In order to define good policies on how to detect and to react on problems in the operation of a distributed applications, knowledge on the topology and the dependencies of the different components is necessary. In [7], an event correlator based on dependency graphs is introduced. Using this dependency graph, it is possible to identify which components of a distributed system will be affected if an error occurs. The advantage of this approach is that it is not limited to react on situations that have been discovered in the past. As the errors are tracked down to single components, the mechanisms to solve a discovered problem are likely less complex and the complicated part of case database oriented systems matching the current problem situation with a stored solution does not apply.

2.4.3 Prediction Systems

Another component needed for autonomous supervision is an intelligent element using historical monitored data as the basis for predicting the behaviour of the system in the near future in order to allow supervision components to react in a proactive way. Significant work in this area exists for prediction of network behaviour e.g. the Network Weather Service (NWS) (see [8]) or the Remos System (see [9]). The common feature in these toolkits is to collect data on the supervised network and use this historical information in order to predict the load of the network in the near future. However they are almost limited to the network layer with a small part of monitoring of the system status.

3. INTRODUCTION TO GENESYS

The following sections are intended to present an innovative solution for distributed systems supervision. Through this example, the reader will be able to understand the real and practical issues of designing, specifying, implementing a generic, open and comprehensive supervision solution.

3.1 What is GeneSyS ?

GeneSyS (Generic System Supervision) is a European Union project (IST-2001-34162) co-funded by the Commission of the European Communities (5th Framework). EADS SPACE Transportation (France) is the project Co-ordinator, with University of Stuttgart (Germany), MTA SZTAKI (Hungary), NAVUS GmbH and D-3-Group GmbH (both of Germany) as participants. GeneSyS started in March 2002 with planned completion in October 2004 [10]. The project is aimed at developing a new, open, generic and modular middleware for distributed systems supervision. Besides, the consortium intends to make GeneSyS an open standard in the distributed system supervision domain.

3.2 Contexts

This section presents three contexts that were chosen as validation scenarios for the project purposes. These quite different application coming from different industrial domains helped to identify a list of requirements that needed to be fulfiled by GeneSyS (see [11]).

3.2.1 Preliminary Design Review

This scenario was brought by EADS SPACE Transportation, the European aerospace industry leader. It concerns the spacecraft production process and, in particular, Automated Transfer Vehicle (ATV) design. The Preliminary Design Review (PDR) is a design process stage, involving hundreds of engineers from different European countries, that meet regularly to discuss ATV technical documentation, to release comments and change proposals.

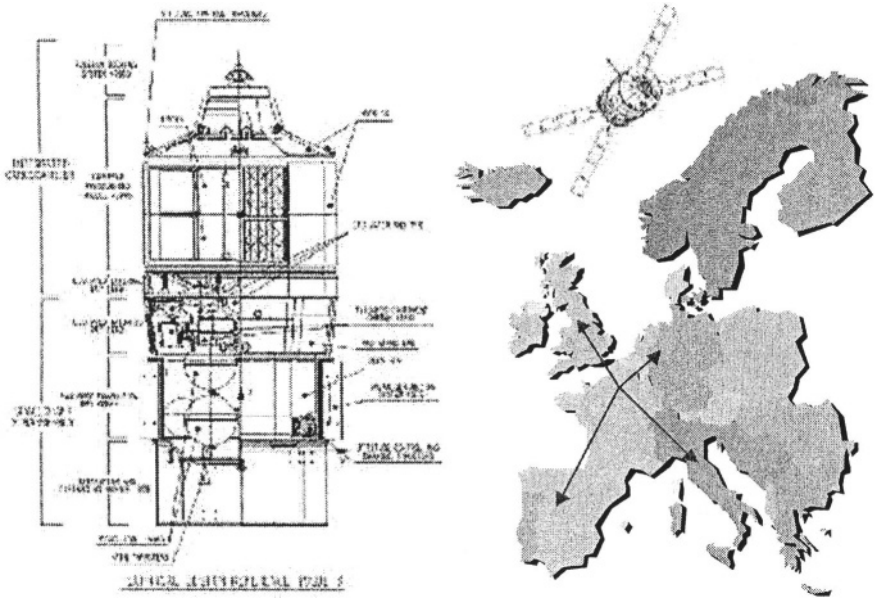

Figure 3. Preliminary Design Review, an ATV design phase.

To reduce the travel costs, a groupware application is used, which allows collaborative work on the documentation and visio conference meetings to discuss comments and change proposals.

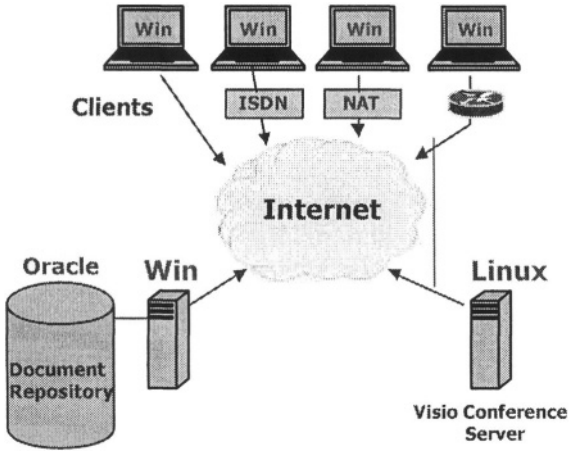

Figure 4. PDR Application, Groupware Application for Collaborative Engineering

The groupware application comprises mainly a Document Repository and a Visio Conference Server supporting multiple simultaneous client access. The application is physically highly distributed, different operating systems and access means are used. In addition, a database management system with a web front-end and the visio conference server require specific supervision on the application level.

Thus the system maintenance and the client technical support seems to be extremely difficult without common generic supervision framework.

3.2.2 Distributed Training

The following scenario is also from the space domain and concerns HLA-based simulations. HLA (see [12]) is a DoD standard for real-time interactive simulations. This standard is widely used in military, aerospace and automotive industries. The Distributed Training Scenario involves 4 real-time simulators playing different roles in joint training sessions of astronauts and ground controllers in order to prepare them in advance for contingency situation during the ATV to International Space Station (ISS) approach manoeuvre.

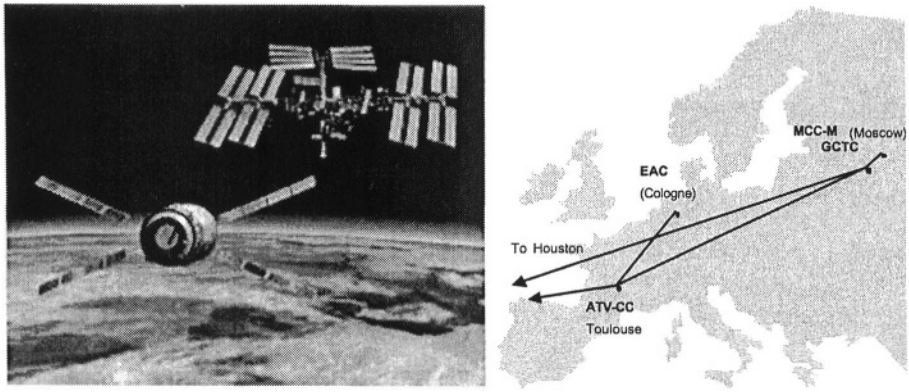

Figure 5. Distributed Training - HLA-based Interactive Simulation

The trainee teams are located in different places all over the world (Toulouse, Houston, Moscow), which imposes performance constraints on a supervision solution.

The supervision is needed at all levels starting from operating system, up to HLA middleware and Training application.

3.2.3 Web-Servers Monitoring

Today's web servers implement a distributed, multi layered architecture hosting complex applications. The situation is even complex if such web applications are connected to each other requiring continuous synchronisation with each other. In the Web Servers monitoring scenario, the supervision of such complex web based applications are contemplated. One such application is the so-called "node server" of the StreamOnTheFly (SOTF) application. SOTF provides a peer-to-peer network for community radios to share their shows (see figure 6).

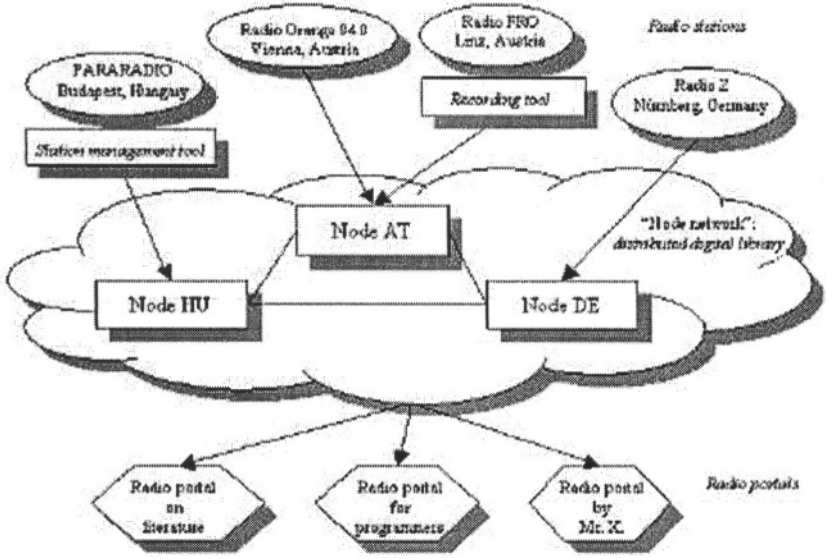

Figure 6. Web Servers Monitoring Scenario - Stream on the Fly Application

Nodes are the repositories of the shows, collecting the audio files and their associated metadata. The metadata is periodically exchanged by the highly distributed nodes of the SOTF network. Each node itself is typically a distributed system as the HTTP server and database servers are typically located on different machines or in different domain, therefore not only the collaboration of these nodes but the operation of a single node requires a complex supervision solution.

3.3 Requirements

Analysing the need of supervision for the mentioned distributed applications, the following common requirements were identified:

Comprehensiveness - the supervision should be provided for all levels of a computer infrastructure.

Flexibility - all kinds of data types and complex data structures should be supported.

Portability - the supervision system should by compatible with distributed applications, that are often multi-platform, involving different access means.

Integration capability - a global supervision system should be capable to benefit from existing solutions for local supervision and from available application extension mechanisms.

Security - authorisation, authentication, data integrity and privacy are extremely important for the distributed systems with control functionality.

4. GENESYS FRAMEWORK

4.1 Constraints of Existing Solutions

The commercial supervision systems like Tivoli, OpenView, Unicenter TNG, mentioned earlier, are aimed at different aspects of system monitoring starting from operating systems to network and up to some commercial standard applications.

However most of them have several common constraints, which should be overcome:

- proprietary interfaces;
- proprietary protocols;
- operating system dependent implementation;
- non-flexible architecture;
- dedication to particular commercial application (Oracle, SAP, etc.).

Although these supervision systems use open standards (SNMP, JMX, Corba), the mentioned constraints complicate integration with third-party monitoring tools to achieve system control at all levels. At the same time, proprietary solutions slow down pace of development of the whole domain.

With the advancement of Web technologies, more and more works appeared to introduce these technologies in the world of supervision (DMTF WBEM, OASIS WSDM, etc.). GeneSyS was one of the first to bring the Web Services to this domain. Besides, the GeneSyS consortium intends to make the GeneSyS achievements a new open standard.

4.2 Design Objectives

Thus, the main GeneSyS objective is to design a system supervision middleware that can be used in a wide range of applications (examples are listed in the applicability section). The planned outcome of the specification phase was a communication and messaging API for the middleware components as well as the functional design of these components.

While designing the GeneSyS framework, the consortium continuously aligned the design to meet the following aspects of a new solution in order to fit the requirements:

- the framework must clearly separate supervision (collection and processing of data) from visualisation (display and analysis of data)
- the framework must support both passive monitoring (collection of runtime data) and active control (start, stop, reconfiguration) of monitored entities
- the framework must provide all functionality related to the communication between middleware components
- the communication protocol must provide secure message exchange between middleware components
- the framework must be based on open standards and protocols to assure its openness and easy adaptation by anyone in need of a supervision facility; dependency on third party or proprietary software is not permissible
- the specified API must be language and implementation neutral
- authors of new monitoring components should only bother with the details of how to get the monitoring data from a monitored entity and how to control it - the rest (transferring the data to other components, storing the data, querying historical data, visualising monitoring data, etc.) should all be handled by GeneSyS.

4.3 Web Technologies as a Platform for a Supervision Framework

As the result of our research matched with the requirements outlined for GeneSyS, an agent based approach was implemented which separates the monitoring/controlling and visualisation of monitoring data. Web Services technologies were chosen as the base for GeneSyS messaging protocol.

Basing the supervision infrastructure on agents seems logical, because the monitoring of IT entities requires properties that are available with software agents. A software agent is a program that is authorised to act for another program or human (see [13]). Agents possess the characteristics of delegacy, competency and amenability that are the exact properties needed for a monitoring software component.

Delegacy for software agents centres on persistence. Delegacy provides the base for an agent to be an autonomous software component, which can act without the intervention of other programs or human operators. “Fire-and-forget” software agents stay resident, or persistent, as background processes after being launched. By making decisions and acting on their

environment independently, software agents reduce human workload by generally only interacting with their end-clients when it is time to deliver results. In case of GeneSyS, the agents reside either on the computer hosting the monitored entity or on a computer that is able to communicate with the monitored entity.

Competency within a software environment requires knowledge of the specific communication protocols of the domain (SQL, HTTP, API calls). A monitoring agent competency is to have knowledge about the monitored entity to be able to collect runtime information from it or to control it with commands.

Amenability for non-intelligent software agents is generally limited to providing control options and the generation of status reports that require human review. Such agents often tend to be brittle in the face of a changing environment, necessitating a modification of their programming to restore performance.

Amenability in intelligent software agents can include self-monitoring of achievement toward client goals combined with continuous, online learning to improve performance. GeneSyS makes no restriction on its agents or on their intelligence or autonomous operations, but provides the ability to include it as found necessary by agent writers and also provides some middleware components (like monitoring data repository) that can be used to implement amenability.

Openness and standards based solution was one of the key requirements of GeneSyS especially in the light of the Consortium's intention to turn GeneSyS itself into an industry standard. After a number of iterations, we had two candidates for the realisation of the communication protocol:

- InterAgent Communication Model (ICM - FIPA based) (cf. [14])
- Web Services technologies (see [15])

The ICM framework has not been designed for monitoring or supervision needs but is a general communication framework for inter-agent communication. The Web Services framework standardised by the W3C is a generic framework for the interaction of Services over the Internet and is designed to exploit as much as possible existing protocol frameworks such as SOAP and HTTP. The Web Services framework is in contrast to ICM more a hierarchical or client-sever communication model.

ICM is a very efficient system for the transmission of messages between agents. However for supervision in general and especially in the area of standardisation and flexibility, major disadvantages have been identified. The most important issue against ICM was that the GeneSyS Message format would be tight to the ICM communication protocol and would bring GeneSyS in a complete dependency to ICM. This is a serious risk as ICM is

not used at all by important software companies and no activity in the development has been identified since autumn 2001.

Web Services has a major problem with respect to performance. The use of an XML based protocol cannot be as efficient as a binary protocol due to the consuming text processing. Additionally, the most common transport protocol used for SOAP messages, the Hypertext Transfer Protocol (HTTP), is not very efficient as it lacks stateful connections. However we are convinced that these problems can be solved as Web Services potentially can use different protocols. The feature of alternative protocol bindings is already used for example in the .NET framework using Remoting, which uses different (proprietary) protocols. As this problem is not solely part of GeneSyS but the whole community including the major software vendors that are committed to Web Services will face this problem, the assumption that this limitation will disappear seems reasonable.

After a detailed comparison of these two technologies, we selected Web Services, including the SOAP XML based communication protocol as a base for GeneSyS. Going on the Web Services path, we have a strong industry backing with tools available for many languages. With this decision, we also defined the first instance of a Web Services based supervision system that has recently been followed by other companies and standards organisations (OASIS WSDM, DataPower Technology [16])

On top of SOAP and Web Services, a new layer of the GeneSyS protocol has been established called the GeneSyS Messaging Protocol (GMP). GeneSyS Messaging Protocol is a lightweight messaging protocol for exchanging structured supervision information in a decentralised, distributed environment. It is an XML protocol based on XML 1.0, XML Schema and XML Namespaces. GMP is intended to be used in the Web Services Architecture, thus, SOAP is considered as a default underlying protocol. However, other protocol bindings can be equally applied. Using XML to represent monitoring data was a natural choice. XML is a widely accepted industry standard that supports structured representation of complex data types, structures (enumerations, arrays, lists, hash maps, choices, sequences) and it can be easily processed by both humans and computers. With the wide acceptance of XML, an integration with supervised application and 3d party monitoring solutions can be smoothly achieved, since XML toolkits are available for every platform.

4.4 Basic Components and Communication Model

This section provides implementation details, illustrating a common supervision framework architecture.

Fig.7 depicts the basic GeneSyS functionality.

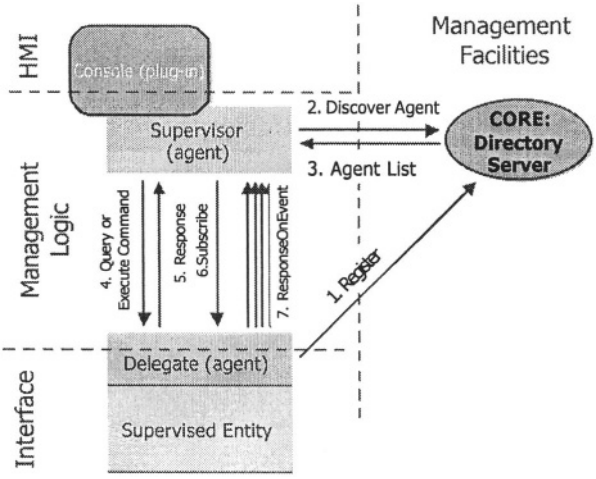

Figure 7. GeneSyS Communication Model

As showed above, supervision process involves several generic components. The Delegate implements an interface to the Supervised Entity (Operating System, Network, Applications, etc.), retrieves and evaluates monitoring information and generates monitoring events. The Supervisor is a remote controller entity that communicates with one or more Delegates. It may encapsulate management automation functionality (intelligence), recognising state patterns and making recovery actions. The Console is connected to one or many Supervisors to visualise the monitoring information in a synthetic way, and to allow for efficient controlling of Supervised Entity. The Core implements Directory Server, a location storage being updated dynamically.

The agents register to the Core to make them discoverable by other agents. Hereafter, the "agent" is a generic term comprising the Supervisor and the Delegate.

Both "pull" and "push" interaction models are available. The pull model is realised by the Query/Response mechanism, while the Event Subscribe mechanism secures the push model. All interactions between agents are

provided for by the SOAP-RPC. The flexibility of XML standard is used to encode communication messages (GeneSyS Messaging Protocol) supporting complex data structures and custom data types.

4.5 Intelligence

An inherent property of software agents is autonomy, that is, the ability to work without the intervention of other programs or humans. Autonomous work requires some level of intelligence so that the agent can react on changes in its environment or can make decision based on its internal logic driven by rules or other means. Intelligence in agents is also required because in a complex environment with some 10 or 100 monitored entities, an administrator could be easily flooded with low level warnings like "memory is running low" or "maximum number of users almost reached". Instead, the administrator first needs a general, summarised view about the health of the systems and then can look at the details as necessary.

GeneSyS agents can work autonomously in a hosting environment connected to a monitored entity from which it collects data or controls its operation. The GeneSyS framework provides API hooks for adding intelligence to agents as well as components for supporting the implementation of intelligence. Intelligence can be accomplished in several ways, that are only outlined here, as the actual implementation of this feature is not a main goal of GeneSyS:

- Specific Implementation: the „intelligence" to react on the system status can be done as part of the program code of the agent.
- Parameter based Generic Solution. The rules can be configured through parameters. A basic example is a „Threshold Miss Agent" where the parameters would be min and max values.
- Rule Based Systems. In complex settings, the usage of rule based systems could be an option where the rules can be expressed in an external file e.g. based on JESS.
- Workflow based systems. Another option could be to use workflow languages such as BPEL4WS to define workflows that act depending on events receive.

GeneSyS provides a data Repository that is connected to the middleware bus via the same API as any other agents, which means its functionality is available to all other agents connected to a given CORE. The Repository provides a generic XML data storage facility. Agents can store monitoring or control messages in the Repository, which can later be queried. With the use of the Repository, an agent can base its decisions on archived data, for example, by analysing past messages for detecting trends in the operation of

the monitored entity. More over, the Repository is also capable for storing control messages – or a list of control messages – which can be "replayed" any number of times at any time it is necessary.

The Agent Dependency Framework (ADF) is another aid for adding intelligence to monitoring. ADF allows defining dependencies of monitored entities. To be more precise, not directly the dependencies of monitored entities but the dependencies of the agents monitoring the entity can be described. Each delegate agent can describe in its component description (which is stored in the CORE) what agents it depends on. The dependency forms a directed graph that should never cause a circular reference. Once the dependency of each delegate is described, the dependency graph can be queried from the CORE. Based on the dependency graph, a special supervisor console view can be created that draws a tree view of the dependent entities and gives a quick overview of the health of the system with green, yellow and red light depicting a healthy, questionable or erroneous state of the dependant systems. This way of visualising the monitored system with all its dependent components provides a way for tracking root cause of problems. For example, an administrator seeing a red light in the top of the dependency hierarchy can expand the tree until he finds the subsystem that generates the red light and which has been "propagated" up in the dependency tree. In the same way, an autonomous intelligent agent can walk this tree and find the root cause of the problem and can work only with that subsystem that was the source of the problem.

5. APPLICABILITY RESULTS

This section presents the applicability results in accordance with the industrial contexts in order to give real examples of the GeneSyS framework in use.

5.1 Preliminary Design Review Scenario

This scenario was intended to prove a viability of the GeneSyS concept. Common system and network agents were developed to reflect system administrator needs. Custom application agents were used to monitor the system functional status (application load, resources used by applications, etc), user activities (documentation in use, on-line meetings, access violation, etc). The figure 8 depicts the deployment of GeneSyS components involved in the scenario.

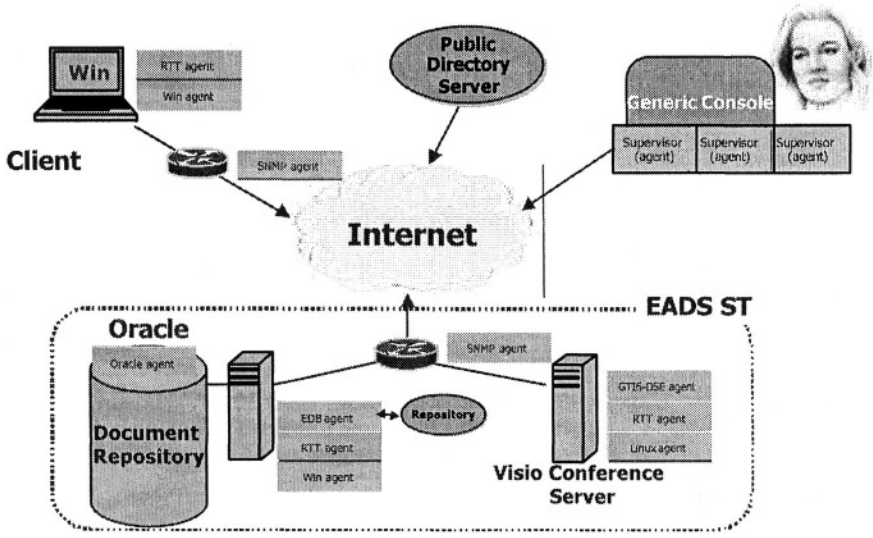

Figure 8. The PDR Application Supervision

The main goals for this scenario was to prove a capability of Web Services based distributed system to work in an heterogeneous environment. It includes support of different operating systems (Windows, Linux), programming languages and toolkits(C/C++/gSOAP, Java/Axis, .Net). Besides, developing custom application agents (Oracle, EDB, GTI6-DSE, Mbone, Tomcat), the integration capability was ensured.

The validation showed that, besides some ergonomy and performance issues, the solution is ready for the large community of the internet users. That is why, generic components for system and network monitoring, as well as, visualisation tools, service components and development toolkits were released under open source policy and can be found at the GeneSyS SourceForge repository (see [17]).

5.2 Distributed Training Scenario

The Distributed Training scenario was implemented in order to improve the GeneSyS V1 functionality and usability as well as to introduce an intelligence basis.

The flexible GeneSyS information allowed customising of System and Network agents and development of scenario specific Middleware and Application agents (RTI middleware, DIS-RVM application).

Figure 9 depicts the deployment schema and gives intelligence implementation hints.

The "synthetic view" and "agent dependencies framework" approaches were used to provide administrators with a run-time system operation status summary and to allow a fast problem location.

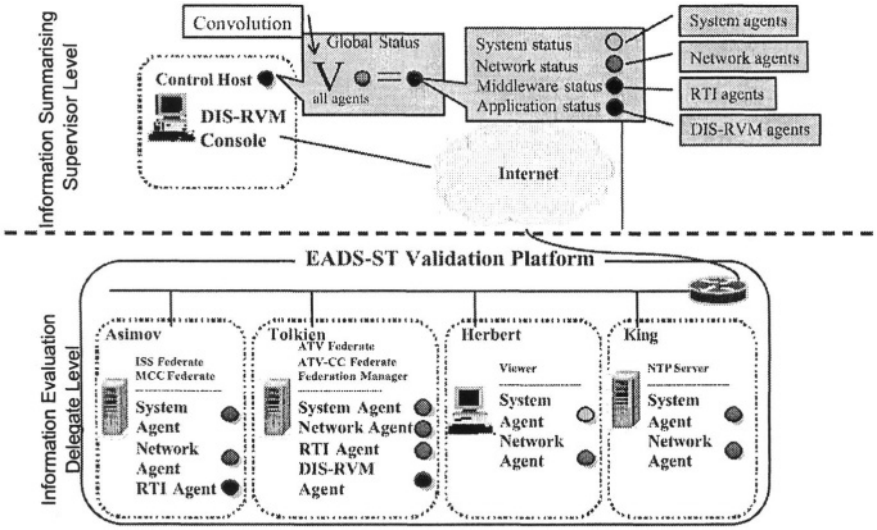

Figure 9. Intelligence in Distributed Training Supervision

Thus an administrator could browse down the agents to find a problem origin and then maintain the system.

5.3 Web Servers Scenario

The Web Servers Monitoring validation scenario aims at using GeneSyS for monitoring and controlling web servers and web based on-line services.

A Web Server is typically more than just an HTTP daemon: it may invoke external programs and those programs may use other programs for their execution, and so on. A typical Web Server can include, for example, an Apache server with a PHP interpreter and a MySQL database used by a number of PHP application. The Web Server is considered "healthy" only if all of these components are in good condition. Because these components may be dependent of each other it is not enough to have separate agents for all entities but these agents must be connected in a way to reflect the dependencies of the monitored entities.

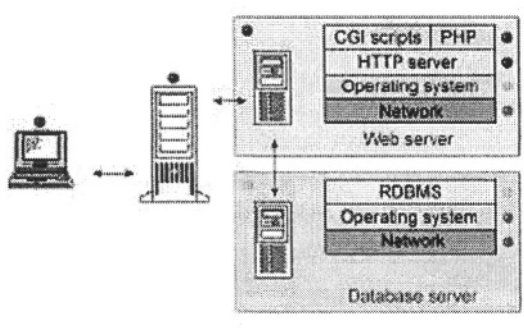

Figure 10. Web Application - A Common Deployment

Going on with the previous example: a Web Server could be considered healthy if the Apache daemon is up and running, the PHP applications it hosts respond in an acceptable time interval and the MySQL server has enough space for new records. If any of these conditions are not met the system should notify the administrator. More over, the unresponsiveness of Apache may be the result of a number of other dependent subsystems, like the operating or network system. So the "monitoring entity" could be divided into some more elements, namely the Apache server itself, the underlying operating system and the network connecting the server machine to the outer world. In this case even if Apache is found to be alive the operating system agent may report that the CPU load is too high and this could cause in a short time the Apache server being unable to respond to requests.

The Web Servers Monitoring scenario extensively uses the Agent Dependency Framework of GeneSyS, which provides the ability to describe the dependencies of system components and use this dependency graph to detect and find root cause of an erroneous system state.

6. CONCLUSION

Today, the distributed systems supervision is a very complex and important issue. The main purpose of this chapter was to give some useful information :

- To understand the problematic of this specific supervision,
- To know the existing technologies, tools in this domain,
- To see what are the main innovative features needed,
- To illustrate, through a case study - the GeneSyS project, the real-life needs, the architecture and design of a generic supervision solution.

In comparison with other solutions, among other advantages, the authors would like to emphasise that the GeneSyS architecture is open to be extended with custom agents for all kind of applications. Besides the proposed framework is published on SourceForge repository under an open source policy and already available for deploying (see [17]).

REFERENCIES

[1] JSR-000048 WBEM Services Specification, Sun Microsystems, Inc. October, 2002

[2] Hatonen, K. ; Klemettinen, M. ; H., Mannila: Knowledge discovery from telecommunication network alarm databases. In: International Conference on Data Engineering (ICDE'96), 1996, S. 115–122

[3] LEWIS, L.: A case-based reasoning approach to the resolution of faults in communication networks. In: Integrated Network Management III, 1993, S. 671–682

[4] RODOSEK, G. D. A Framework for Supporting Fault Diagnosis in Integrated Network and Systems Management: Methodologies for the Correlation of Trouble Tickets and Access to Problem-Solving Expertise. 1995

[5] R. Gardner and D. Harle. Pattern discovery and specification translation for alarm correlation. In Proceedings of Network Operations and Management Symposium (NOMS'98), New Orleans, USA, February 1998, pages 713–722.

[6] D. Ohsie, A. Mayer, S. Kliger, et al. Event modeling with the model language. In A. Lazar, R. Saracco, and R. Stadler, editors. Integrated Network Management V (IM'97), San Diego, USA, May 1997. Chapman & Hall, pages 625–637.

[7] GRUSCHKE, Boris: Integrated Event Management: Event Correlation using Dependency Graphs. In: Proceedings of DSOM'98, 1998

[8] Rich Wolski et.al., The Network Weather Service

[9] A. DeWitt, T. Gross, B. Lowekamp, N. Miller, P. Steenkiste, J. Subhlok, D. Sutherland, *"ReMoS: A Resource Monitoring System for Network-Aware Applications"* Carnegie Mellon School of Computer Science, CMU-CS-97-194.

[10] GeneSyS project official web-site : http://genesys.sztaki.hu

[11] GeneSyS V2 User Requirements Document - D1.2.1

[12] Institute of Electrical and Electronic Engineers - IEEE 1516.1, IEEE 1516.2, IEEE 1516.3

[13] Wallace Croft, David, "Intelligent Software Agents: Definitions and Applications", 1997, http://www.alumni.caltech.edu/~croft/research/agent/definition

[14] The Inter-Agent Communication Model (ICM), Fujitsu Laboratories of America, Inc., http://www.nar.fujitsulabs.com/icm/about.html

[15] Web Service Activity of W3C, http://www.w3.org/2002/ws/

[16] DataPower Offering Web Services-Based Network Device Management, http://www.ebizq.net/news/2534.html

[17] GeneSyS project SourceForge file repository, http://www.sourceforge.net/projects/genesys-mw