Thèse de Doctorat de l'Université Paris VI Pierre et Marie Curie

Spécialité : **Systèmes Informatiques**

présentée par : **Andrey SADOVYKH**

Pour obtenir le grade de Docteur de l'Université Paris VI Pierre et Marie Curie

Sujet de la Thèse :

Concept innovateur d'un middleware pour la supervision de systèmes complexes.

Titre en Anglais :

Innovative Middleware Concept for Supervision of Complex Systems

Soutenue **le 6 avril 2005** devant le jury composé de :

Directeurs de Thèse :

Serge FDIDA	Prof. à l'Université Pierre et Marie Curie
Antoine LAYDIER	EADS SPACE Transportation

Rapporteurs :

Michel DIAZ	Dir. de Rech. au CNRS, LAAS-CNRS
Olivier FESTOR	Dir. de Rech. au LORIA-INRIA Lorraine

Examinateurs :

Ramon PUIGJANER	Prof. à l'Universitat de les Illes Balears
Stefan WESNER	HLRS (High Performance Computing Center), Stuttgart
Marie-Pierre GERVAIS	Prof. à l'Université Paris X Nanterre

CLASSIFICATION DU DOCUMENT	EADS SPACE TRANSPORTATION	CONFIDENTIEL EADS-ST NON PROTEGE

RAPPORT DE STAGE

RENSEIGNEMENTS CONCERNANT LE STAGE

Sujet	:	Concept innovateur d'un middleware pour la supervision de systèmes complexes
Dates	:	01/10/2002-06/04/2005
Unité ou service d'accueil	:	TE641
Maître de stage	:	Antoine LAYDIER

RENSEIGNEMENTS CONCERNANT LE STAGIAIRE

Nom	:	Andrey SADOVYKH
Ecole	:	LIP6
Niveau d'études	:	BAC+6
Spécialité	:	Systèmes Informatiques
Responsable de stage de l'école	:	Serge FDIDA
Date et lieu de soutenance	:	le 06 avril 2005 au LIP6, Paris

VISAS DU RAPPORT

MAITRE DE STAGE	PROPRIETE INDUSTRIELLE	SECURITE INDUSTRIELLE	SERVICE REPROGRAPHIE
Ce rapport est-il confidentiel : OUI ☐ NON ☐			
DATE - VISA	DATE - VISA	DATE - VISA	DATE - VISA

DIFFUSION : HF2

Ce travail a été réalisé dans le cadre du projet Européen GeneSyS (Generic Systems Supervision) IST-2001-34162

J'aimerais exprimer ma gratitude envers tous les membres du jury de cette thèse :

Serge FDIDA et **Antoine LAYDIER** d'avoir dirigé mes travaux de thèse.

Michel DIAZ et **Olivier FESTOR** pour l'honneur qu'ils m'ont fait d'avoir accepté d'être rapporteurs de cette thèse.

Ramon PUIGJANER, **Stefan WESNER** et **Marie-Pierre GERVAIS** d'avoir accepté d'être membre du Jury de la thèse.

Remerciement spécial à **Jean-Eric BOHDANOWICZ**, chef du projet GeneSyS, qui a dirigé mes travaux au sein de EADS SPACE Transportation et qui m'a beaucoup aidé à rédiger le texte en français, pour ses remarques et commentaires.

Je tiens aussi à remercier **Daniel CLAUDE** pour l'honneur qu'il m'a fait de m'avoir invité à travailler à EADS et pour son aide à la relecture du document.

Je remercie également **Alexander VANKOV** et **Petr SHLYAEV** de D-3-GROUP pour leur soutien. Je remercie également tous mes collègues d'EADS et du LIP6 pour m'avoir accueilli dans leurs équipes.

Je remercie tous les membres de l'équipe GeneSyS pour le plaisir d'avoir travaillé avec eux.

Je remercie ma femme, **Svetlana**, qui m'a soutenu pendant ces trois années et qui, de plus, m'a beaucoup aidé à corriger la stylistique anglaise.

Enfin, je remercie mes parents pour leur attention et leur présence.

Résumé :

Durant les dernières années, l’informatique passe des systèmes monolithiques vers des systèmes répartis multi-modulaires. Ces systèmes exigent une solution appropriée pour leur supervision aussi bien au niveau du réseau et que de l'application. L’état de l’art au niveau des standards de systèmes commerciaux de la supervision montre que les solution existantes partagent les contraintes communes suivantes : interopérabilité, portabilité, flexibilité, un manque de solutions complètes…

La thèse propose une architecture ouverte, modulaire et flexible basée sur les Web Services. L’innovation principale repose sur une approche de conception dédiée à optimiser l’intégration de tous les moyens de supervision dans un système unique et puissant.

Les travaux de recherches ont été effectués en collaboration forte avec des industriels afin de valider le concept sur les systèmes réels. Les résultats obtenus sont significatifs, car représentatifs pour la plupart de systèmes repartis.

Mots Clés :

Systèmes de Supervision, Middleware de Supervision, Web Services, GeneSyS.

Abstracts:

During the past years the IT domain showed a continuous trend to move from a single computer towards complex networks and from monolithic systems to multi-modular distributed ones. The distributed systems require an appropriate supervision solution both at network and application levels. The state of the art research in distributed management standards and commercial frameworks showed that existing solutions have such common constraints as interoperability, portability, flexibility and a lack of comprehensive solutions.

This thesis proposes an open, modular and flexible architecture based on Web Services. The main innovation is the design approach purposed to optimise integration of all available supervision means into a unique powerful system.

The work included a strong cooperation with industrial partners in order to validate the concept on real systems. The obtained results are significant, since they are representative for most distributed systems.

Keywords:

Supervision Systems, Supervision Middleware, Web Services, GeneSyS.

Résumé

1. INTRODUCTION

1.1. Définition du Problème

Durant les dernières années, les technologies de l'information montrent une tendance à passer de l’ordinateur isolé vers des réseaux complexes et des systèmes monolithiques vers des systèmes répartis multi-modulaires.

Les systèmes répartis exigent une solution appropriée pour leur supervision aussi bien au niveau du réseau et que de l'application. Pour gérer ce problème, il existe plusieurs standards et systèmes de supervision, comme, par exemple, SNMP [9], JMX [10], Tivoli (IBM) [11], OpenView (HP) [12] et NAGIOS [13]. Ils visent différents aspects de la supervision, allant des systèmes d'exploitation, en passant par le réseau, jusqu’à certaines applications commerciales courantes.

Toutefois, la plupart d'entre eux affichent plusieurs contraintes communes, certaines d'entre elles sont indiquées dans la liste ci-dessous :

- Interopérabilité : Des composants écrits en différents langages de programmation et utilisant différentes implémentations sont censés utiliser les mêmes spécifications d’architecture mais peuvent ne pas être capables de coopérer à 100%.
- Portabilité des composants : Souvent, les composants sont conçus pour fonctionner uniquement sous leur système d'exploitation d’origine comme Windows ou Linux. Ils sont très dépendants des mécanismes de transport et d'une manière générale des protocoles de communication de bas niveaux propres à leur système d’exploitation d’origine.
- Complexité de développement/déploiement : Beaucoup d'applications commerciales reposent sur des interfaces de programmation (API) propriétaires qui rendent difficile la création de nouveaux agents et la connexion de ceux-ci à un seul système de supervision existant.

- Architecture non-flexible: Lorsque l'agent de supervision et l'outil de visualisation sont implémentés dans le même composant, alors les mises à jour de la console impactent les fonctionnalités de l'agent et vice versa.
- Manque de solutions complètes : Il existent de nombreuses solutions différentes ne permettant de superviser que des éléments précis d'un système. Pourtant, une solution complète de supervision permettrait d'évaluer l'état global du système surveillé. Par exemple, l'état d'une application dépend fortement de l'état du système d'exploitation qui la supporte. A partir du moment où la surveillance d'une simple application ne peut donner une vue correcte de l'ensemble des problèmes possibles, il devient nécessaire d'intégrer les informations en provenance de tous les composants impliqués dans le système.

Les contraintes mentionnées ci-dessus compliquent l'intégration d'outils de surveillance en provenance de tiers. Cette intégration est essentielle pour assurer le contrôle du système à tous les niveaux. Par ailleurs, les solutions propriétaires freinent le développement de ce domaine.

Par conséquent, l'objectif de mes travaux de recherche est d'étudier un concept de middleware générique, modulaire et complet de supervision, qui permettrait de résoudre ou de réduire les limitations indiquées ci-dessus.

1.2. Contributions

Ces recherches doctorales ont été effectuées sous la direction du Laboratoire d'Informatique de l'Université Paris 6 dans le cadre du projet GeneSyS au sein de la société EADS SPACE Transportation.

Les contributions de l'auteur sont brièvement récapitulées ci-dessous :

- Une étude de deux systèmes repartis industriels dans le domaine aérospatial (une application de travail collaboratif et une simulation distribuée interactive) afin de déterminer des exigences communes pour leur supervision ;
- Analyse de l'état de l'art dans le domaine des technologies et systèmes de supervision distribuée afin d'étudier leur applicabilité aux systèmes industriels étudiés ;
- Analyse des technologies utilisées dans les différents progiciels pour limiter les solutions possibles ;

- Etude et spécification d'un système de supervision basé sur la technologie des Web-Services.
- Conception et implémentation de plusieurs prototypes destinés à la supervision des systèmes repartis industriels étudiés : Application d'Ingénierie Collaborative et Simulation Distribuée Interactive;
- Démonstration de l'adéquation de ces prototypes aux besoins utilisateurs au moyen de deux scénarios de validation ;
- Évaluation des caractéristiques de la solution implémentée.

1.3. Méthodologie et sommaire du document de thèse

Les travaux menés dans le cadre de cette thèse et énumérés dans la section précédente sont décrits dans les différents chapitres de ce document:

- Chapitre 2 : La section « contexte du problème » donne les principales définitions des concepts utilisés dans le document: Systèmes Complexes, Systèmes Distribués, Systèmes de Supervision. Ces définitions sont illustrées avec des exemples réels : « Preliminary Design Review » (Revue de Design Préliminaire - système de travail collaboratif) et « Distributed Training Scenario » (Scénario d'Entrainement Distribué - simulation interactive de la mission de Rendez-Vous de l'ATV et de l'ISS). Le but de ces exemples est de fournir des exigences représentatives des besoins des utilisateurs d'un système de supervision dans ces différents contextes qui serviront de critères d'évaluation lors de l'analyse du produit final;
- Chapitre 3 : La section « état de l'art » décrit des technologies et des systèmes existants, ainsi que des projets scientifiques, dans le domaine de la supervision de systèmes distribués. Le chapitre finit par une analyse des solutions existantes et de leur applicabilité aux problèmes industriels mentionnés ci-dessus;
- Chapitre 4: Les sections de « conception et d'implémentation » présentent les points importants qui ont été analysés lors de la phase de conception et se concentrent sur les innovations liées à ce genre d'outil de supervision et aux technologies associées. Le chapitre commence par une hypothèse : la technologie des Web-Services pourrait être un bon choix d'implémentation pour un outil de supervision. Une analyse des technologies utilisées dans les progiciels disponibles permet de justifier

cette hypothèse. Ce chapitre se poursuit par une présentation détaillée de la technologie des Web-Services afin d'aider le lecteur dans sa compréhension de l'approche choisie par l'auteur. La section « Proposed Architecture » décrit une proposition d'architecture permettant de réaliser un système de supervision en se basant sur les Web-Services. Enfin, la section « Prototype » décrit les différentes approches et solutions qui ont été implémentées afin de superviser les systèmes industriels étudiés ;

- Chapitre 5: la section « d'évaluation » discute des limitations de la solution implémentée en se basant tout d'abord sur les résultats des tests de performance effectués puis sur les étapes de validation qui ont fait participer des utilisateurs réels. Enfin, la conformité de la solution vis-à-vis des besoins exprimés est étudiée.
- En conclusion, le document résume tous les résultats majeurs acquis dans le cadre de cette thèse et indique un certain nombre d'axes potentiels pour de futures recherches.

2. CONTEXT DU PROBLEME

2.1. Définitions

Le but de la première partie du chapitre est de définir les principales notions utilisées dans le document :

- supervision – fonction de surveillance et de gestion de tous les aspects opérationnels d'un système complexe, y compris : disponibilité des applications, systèmes d'exploitation, équipement réseaux, flux de données, stockage de données, activités des utilisateurs, droits d'accès, etc.
- système distribué – un sous-ensemble de systèmes complexes, qui a particulièrement été étudié dans la thèse. La complexité de ces systèmes se caractérise par un grand nombre de composants et de relations entre ces composants, par l'hétérogénéité des types de composants et des moyens d'accès, ainsi que par l'étendue de la distribution ;
- systèmes de supervision distribuée – ces systèmes sont destinés à la supervision de systèmes repartis et sont eux mêmes des systèmes distribués. Une architecture classique de supervision distribuée sous-entend une utilisation de composants individuels pour chacun des aspects d'un système repartis, ainsi que l'utilisation d'une application centrale pour la visualisation des statistiques de surveillance et pour l'administration du fonctionnement du système. Cette approche est connue comme « la gestion par délégation », car les fonctions de supervision sont « déléguées » aux composants repartis. Ainsi, les composants sont classés d'après leurs fonctions comme des « délégués » (delegates) et/ou des « superviseurs » (supervisors). La communication entre ces composants est imposée par leurs modes de fonctionnement et leurs fonctionnalités. Par exemple, une communication périodique ou événementielle peut être exigée ;
- un middleware de supervision – un progiciel qui permet de faire le lien entre des sources ou des moyens de supervision et des applications utilisatrices des fonctions d'administration. Le middleware sert à cacher la complexité de la distribution, de la réalisation d'une communication vis-à-vis des utilisateurs finaux. Il est conçu pour fournir une base de conception d'agents de supervision et pour simplifier leur intégration avec des outils

de visualisation. L'étude de ces nouveaux middlewares de supervision est le thème principal de ces travaux de recherches.

2.2. Problèmes Industriels Réels

La deuxième partie du chapitre illustre les définitions au moyen de deux applications industrielles distribuées utilisées par EADS SPACE Transportation.

Tout d'abord, l'application PDR (Preliminary Design Review) est une application d'ingénierie collaborative, utilisée à l'échelle européenne pour la conception du véhicule de ravitaillement de la station spatiale internationale, l'ATV (Automated Transfer Vehicle). Cette application inclut des serveurs de documentation et de visio-conférence basés sur les plateformes Windows et Linux. De plus, les applications clientes sont programmées en PHP, Java et C++. La supervision de ce système est nécessaire afin de pouvoir configurer, surveiller et contrôler les éléments suivants:

- l'accès de nombreux clients internationaux ;
- les flux multimédia H.323, T.120 (bande passante, nombre de connexions aux serveurs, etc.) ;
- les équipements réseau ;
- les systèmes d'exploitation concernés (Windows, Linux) ;
- la base de données Oracle ;
- le serveur de visioconférence (MCU) ;
- le serveur d'applications Apache Tomcat;
- l'application permettant d'accéder aux documents ;
- les applications clients.

Le problème est compliqué à cause de l'étendue de la distribution du système global, car il doit être capable de fournir ses services à des utilisateurs situés dans différents pays européens (France, Allemagne, Angleterre, Italie, Espagne).
La supervision multi-niveaux de ce type de système nécessite une solution portable, flexible et complète.

Deuxième système industriel visé, l'application DIS-RVM (Distributed Interactive Simulation pour Rendez-Vous Mission) est une simulation distribuée interactive conçue pour l'entraînement du personnel de l'ISS (International Space Station) sur des phases spécifiques de la mission de Rendez-Vous de l'ATV et de l'ISS. La simulation est basée sur le standard HLA (High Level Architecture), qui concerne toutes les phases des simulations : la préparation, l'exécution et la collecte de résultats. Le fonctionnement de la simulation nécessite un bon déroulement et un enchaînement correct de toutes les phases et des nombreuses actions qui les composent, et pourtant, la supervision n'est pas prise en compte par l'architecture HLA. L'approche proposée tente de résoudre ce problème d'une façon générique.

Ces deux applications distribuées issues du domaine spatial ont servi de base aux scénarii de validation qui permettent d'évaluer les prototypes résultant des travaux.

2.3. Spécification de besoins communs

Par ailleurs, l'étude détaillée des besoins de supervision liés à ces deux applications réelles et distinctes permet de définir une liste d'exigences communes à un système générique de supervision vis-à-vis des points suivants :

- interopérabilité – le fait que plusieurs systèmes, qu'ils soient identiques ou radicalement différents, puissent communiquer sans ambiguïté ;
- portabilité – capacité d'être utilisé sous différents systèmes d'exploitation ;
- flexibilité dans le sens d'intégration de informations complexes ;
- support d'agents intelligents pour des solutions élaborées ;
- facilité de développement des composants de supervision ;
- facilité d'utilisation de l'outil.

Ces critères sont utilisés tout au long du document pour comparer des approches existantes et des technologies de middleware afin de concevoir une solution appropriée.

3. L'ETAT DE L'ART

Cette section est destinée à décrire la situation actuelle dans le domaine des progiciels de supervision, leurs architectures et leurs domaines d'application. Pour cela, le chapitre étudie trois grandes catégories : les standards, les systèmes commerciaux et les recherches académiques conduites sur des thèmes similaires.

Les sujets abordés sont les suivants :

- Standards de gestion de réseaux :
    - ICMP – protocole de contrôle des messages Internet;
    - SNMP – protocole simple de gestion de réseau.
- Java Management Extension – un middleware de supervision basé sur Java;
- Standards du groupe DMTF ;
- Systèmes de supervision commerciaux existants :
    - Unicenter TNG (Computer Associated) – un système complexe de supervision basé sur SNMP ;
    - Nagios – un système de surveillance simple, puissant et distribué sous une licence GNU ;
    - OpenView (HP) – un système de supervision à l'échelle d'une entreprise appliquant SNMP;
    - Tivoli (IBM) – un système de supervision à l'échelle d'une entreprise.
- Recherches académiques : des projets universitaires et des travaux publiés lors de conférences spécialisées.

La première partie parcourt les standards de niveaux bas de la supervision – des protocoles de gestion de réseaux (ICMP, SNMP), qui sont beaucoup utilisés dans les systèmes classiques. Le chapitre sur Java Management Extension est consacré au survol d'un standard de middleware, qui a pour but d'introduire des moyens de contrôle dans le monde Java. De plus, la section dédiée aux standards du groupe DMTF cite les principales normes de modèles d'information pour la supervision. La partie sur les systèmes commerciaux de supervision fournit des informations sur les architectures et les approches récentes. Par ailleurs, les recherches académiques présentent les derniers travaux effectués dans le domaine.

Cet état de l'art est suivi par une analyse des avantages et des contraintes de ces systèmes de supervision dans le cadre des domaines fonctionnels visés. Cette analyse permet de justifier la nécessité de la recherche d'une nouvelle solution plus appropriée à la supervision de systèmes complexes hétérogènes. Elle démontre que la flexibilité des systèmes existants est compromise, car la plupart d'entre eux repose sur d'anciennes versions de SNMP. De ce fait, le format des messages utilisés est très strict et ne permet pas d'extension simple, ce que limite fortement l'utilisation de données complexes, structurées ou couplées, nécessaires à une supervision de systèmes plus récents. De plus, leur complexité et l'absence de documentation rendent difficile leur utilisation pour les systèmes visés.

Le grand nombre de travaux académiques dans ce domaine permet également de justifier l'étude et le développement d'un nouveau concept de supervision plus adapté aux systèmes actuels.

4. CONCEPTION ET PROTOTYPAGE

Ce chapitre décrit le principal sujet de recherche de la thèse, l'étude d'un concept d'utilisation de la technologie des Web Services pour la réalisation d'un middleware de supervision.

4.1. L'étude de technologies de middleware

Afin d'identifier une meilleure base pour la conception d'une architecture innovante, il était nécessaire d'étudier les solutions générales d'un middleware.

Les standards étudiés dans ce contexte sont les suivants :

- RPC - un protocole permettant de faire des appels de procédures à distance;
- Les middlewares orientés objets (RMI, DCOM, CORBA) ;
- Les middlewares de composants (CCM, EJB) ;
- Les plateformes multi-agents (FIPA, ICM) ;
- Les middlewares de communication par messages (JMS, Message Queuing) ;
- Web Services (SOAP, Dot Net).

Les meilleurs candidats vis-à-vis des contraintes spécifiées ont alors été identifiés : CORBA et Web Services. Toutes les deux, ces technologies sont portables, interopérables et flexibles grâce à l'utilisation de protocoles du niveau d'application (GIOP et SOAP) et à des langages de définition d'interfaces (IDL et WSDL). Ils offrent également des moyens d'intégration flexibles comme des adaptateurs portables d'objets (POA) pour CORBA et des outils de développement intégrés pour Web Services.

Les principales différences entre ces deux approches sont décrites ci-dessous :

- CORBA est strictement orienté objet, tandis que SOAP (et donc Web Services) n'est pas limité à la technologie objet ;
- CORBA sous-entend un couplage d'applications de type client/serveur, ainsi que la présence d'un composant ORB. En revanche, les Web

Services sont conçus pour fonctionner de manière autonome et sont orientés pour une utilisation dans des systèmes fortement repartis.

- En considérant la possibilité d'avoir des clients légers (agents autonomes de supervision), les Web Services propose plus d'avantages car il est facilement possible de les faire fonctionner sans composants supplémentaires (comme des ORBs, par exemple).
- CORBA fixe la sérialisation d'objets par un ORB et, donc, ne laisse pas le choix pour l'encodage de données. SOAP, lui, permet de customiser toutes les parties des messages, charges utiles comprises. Or, nous considérons cette caractéristique comme extrèmement importante pour l'intégration de différents types de données de surveillance et pour la réalisation de passerelles entre des composants de supervision.

Néanmoins, les différences entre les deux approches sont très subtiles, car CORBA est très mature et robuste. Cependant, nous considérons que les avantages de Web Services sont plus intéressants dans le cadre de composants autonomes tel que des agents de supervision. Par conséquent, nous nous sommes concentrés sur l'étude de l'applicabilité des Web Services comme technologie de base d'un middleware de supervision.

4.2. Architecture

La partie architecture présente le concept d'un middleware de supervision basé sur les Web Services. Les innovations principales sont le couplage de la gestion par délégation et de la gestion événementielle, ainsi que l'application des technologies de type Web Services pour la conception d'un middleware de supervision.

La conception de l'architecture se concentre sur les fonctionnalités d'un middleware : la transmission de données, l'intégration des opérations de supervision, l'adaptation à des moyens de supervision hétérogènes, la découverte de ressources de supervision, ainsi qu'un modèle d'informations générique pour la présentation de charges utiles. Dans ce sens, le concept définit les interfaces des trois composants principaux : le superviseur, le délégué et le service d'annuaire (CORE). De plus, le protocole de communication entre ces composants est spécifié.

La technologie des Web Services a fourni la base pour :

- la « customisation » du transport des messages via le protocole SOAP pour implémenter la gestion par délégation et la gestion événementielle;
- la présentation d'opérations de service (découverte d'agents de supervision) et de supervision (surveillance manuelle et automatique, transmission de commandes de contrôle) comme des Web Services en utilisant le langage WSDL ;
- la définition des messages de services (enregistrement des capacités d'un agent, découverte d'un agent particulier), ainsi que des messages de supervision ;
- la description des agents de supervision (un délégué et un superviseur), ainsi que du service d'annuaire (CORE) en WSDL afin de les présenter comme des Web Services.

En appliquant les Web Services, les fonctions de gestion par délégation ont été réalisées comme des opérations de service du « délégué » - « monitor» pour une surveillance manuel et « perform » pour une transmission de commandes. La gestion événementielle a été implémentée en introduisant l'opération « subscribe » dans le service du « délégué » afin de s'inscrire à un événement et l'opération « accept » du coté du « superviseur » pour la transmission des événements.

Les fonctionnalités de support de gestion sont représentées par le service d'annuaire (CORE) qui stocke les capacités des agents inscrits et permet de localiser un agent particulier à partir de ses caractéristiques. Afin d'exposer ses capacités, un agent doit s'enregistrer dans le CORE. Cette procédure est accessible par l'opération « register » du CORE. Pour découvrir un agent particulier ou un groupe d'agents, l'opération « find » doit être utilisée en précisant les caractéristiques demandées.

Les opérations mentionnées ci-dessus sont des paramètres de messages SOAP. La flexibilité du protocole SOAP a permis d'intégrer les parties destinées au service (tampon de temps, identifiants d'agents en communication, type de message) dans l'entête des messages SOAP. Cela rend possible l'utilisation de toute la partie principale des messages SOAP pour les charges utiles.

Bien que l'encodage des charges utiles ne soit pas à proprement parlé géré dans l'architecture, une approche générique a été définie afin de proposer un exemple d'encodage en langage XML qui permettait, de façon flexible, de présenter tous les types complexes de données : des listes, des énumérations, des données emboîtées contenant plusieurs niveaux, des types composés, etc.

Ces activités de conception ont permis de définir les descriptions en WSDL des composants principaux. En utilisant des compilateurs spécifiques, ces descriptions permettent de créer des squelettes d'agents en plusieurs langages de programmation populaires comme C, C++, Java, PHP, Perl, MS C#, MS FORTRAN, etc. Ensuite, les développeurs peuvent remplir un squelette avec une fonctionnalité de supervision. Cela facilite la mise en application du concept proposé.

4.3. Prototypage

Pendant les travaux de prototypage, plusieurs agents ont été développés pour la supervision de certains paramètres des réseaux utilisés, de la consommation des ressources des systèmes d'exploitation Windows et Linux, d'une base de donnée, d'un serveur d'applications, d'un serveur de visioconférence et d'applications complexes, ainsi que des outils générique de visualisation. Les composants de services développés incluent le CORE et les adaptateurs pour C++, Java et PHP, qui implémentent des opérations de services et servent de « points d'entrée » pour l'implémentation des communications avec un élément supervisé et pour intégration de la charge utile.

Ces premiers éléments ont servi d'éléments de base pour construire des solutions complètes de supervision pour les systèmes industriels mentionnés ci-dessus.

La section « prototypage » du document de thèse décrit les contributions de l'auteur à la conception et à l'implémentation des systèmes de supervision suivants :

- pour le système PDR (Ingénierie Collaborative) : conception d'un système de supervision, réalisation d'agents (l'agent EDB), d'un console et validation du prototype ;
- pour le système DIS-RVM (Simulation Distribuée Interactive) : conception d'un système de supervision, réalisation d'agents (MÄK RTI) et validation du prototype ;
- deux approches de supervision intelligente ;
- l'adaptateur C++ pour la plateforme Linux.

Afin de démontrer la flexibilité du concept et la possibilité offerte d'inclure de « l'intelligence » dans la supervision, la phase du prototypage du système de gestion pour DIS-RVM a inclus le développement de deux approches génériques pour une supervision intelligente :

- Collaboration d'agents pour une visualisation générique : Démonstration de la flexibilité du concept proposé grâce à la réalisation d'un complément du protocole de supervision gérant une description de modes de visualisation. Afin de créer un outil de visualisation générique, il est nécessaire de définir une façon de présenter tous les types de données. Notre approche propose de compléter la description des capacités fournie par chaque agent par des champs indiquant son mode de visualisation. Par exemple, pour une visualisation par des tables, des noms de colonnes et des descriptions de paramètres sont nécessaires;

- Corrélation d'informations de surveillance pour une supervision intelligente : Un prototype de système de supervision permettant une synthèse d'informations afin de détecter rapidement la cause initiale de problèmes en série a été réalisé. Dans les grands systèmes complexes, certains paramètres opérationnels sont liés. Par exemple, le fonctionnement d'une application est lié à la charge du système d'exploitation et, en conséquence, à la consommation de ressources de CPU, de la mémoire vivre, des disques durs. Ainsi, une surcharge du CPU par des processus parallèles peut provoquer un défaut de fonctionnement de l'application. Notre approche a consisté à établir un arbre de dépendances entre les paramètres de supervision. Les agents évaluent un statut des paramètres surveillés par rapport aux critères prédéfinis et envoient une information d'état dans chaque message. Cette information est ensuite directement reportée dans l'arbre de dépendances. Ce qui permet d'impacter des paramètres de plus hauts niveaux.

5. EVALUATION

Ce chapitre présente la mise en application de l'architecture de supervision pour des systèmes réels (PDR et DIS-RVM)., ainsi qu'une étude des performances de la solution.

Les activités d'évaluation décrites se décomposent en trois phases représentant chacune un point de vue différent :

- les leçons tirées du développement : la flexibilité du concept, la facilité de développement et d'utilisation ;
- les essais de performance : évaluation de rapidité, consommation et robustesse d'un prototype développé ;
- la validation d'un prototype par des utilisateurs réels d'un système distribué : démontration d'applicabilité de l'approche proposé à un cas concret.

5.1. Essais de performance

Pendant les essais de performance, les agents basés sur l'adaptateur développés pour le C++ (System, MäK RTI, DIS-RVM) ont été testés avec des procédures génériques sur les plateformes réelles. Le temps de réponse des agents a été mesuré dans deux cas: une charge normale (environ 30 requête par second) et une charge lourde (jusqu'a environ 200 requêtes par second). Le cas de charge normale permet d'évaluer le système dans une situation courante d'utilisation, alors que le cas de charge lourde constitue un test de robustesse. Les paramètres surveillés sont décrits ci-dessous :

- le temps de réponse – un intervalle entre une requête et une réponse ;
- la charge de CPU – un pourcentage d'utilisation de CPU par une tache d'agent;
- la bande passante – un nombre de requêtes servies par seconde ;
- le débit et la charge des réseaux – un nombre de bits passés entre un agent et un logiciel de test.

Les tests ont démontré que le système de supervision se comporte comme prévu. Même en cas de charge lourde, aucune perte de message n'a été détectée. Toutefois, il est nécessaire de souligner, que le coût en terme de bande passante du protocole utilisé était important du fait qu'il est basé sur XML. Cela rends difficile l'application de cette approche à certains systèmes ayant des fortes exigences concernant le débit. Cependant, il a été prouvé que le concept était utilisable pour un grand nombre de problèmes, qui nécessitaient l'intégration de plusieurs moyens de supervision.

5.2. Validation par les utilisateurs

La validation a été décomposée en plusieurs scenarii d'utilisation de systèmes industriels : PDR et DIS-RVM. Ce processus a exigé le déploiement de plateformes de validations sur le site de EADS SPACE Transportation. Cela a permis d'inclure des opérationnels (utilisateurs finaux) dans l'évaluation des prototypes.

Les essais ont démontré l'applicabilité du concept pour des systèmes d'ingénierie collaborative et des systèmes de simulations distribuées interactives du point de vue d'un utilisateur. Les acteurs ont joué les scénarii prédéfinis, pendant lesquels ils ont normalement utilisé les systèmes et ont essayé d'insérer des défauts qui ont été détectés à temps par l'outil de supervision. Néanmoins, ils ont tout de même remarqué des problèmes d'utilisation dans le scénario PDR qui ont été résolus dans le scénario de « Distributed Training » pour le système DIS-RVM. Les utilisateurs ont également évalué l'impact de la supervision sur le temps de réponse global du système, et l'ont jugé acceptable.

En conclusion, les résultats de l'évaluation confirment qu'il est possible d'utiliser les Web Services pour la réalisation d'un middleware de supervision.

6. CONCLUSIONS

Ce chapitre fournit un résumé du travail effectué et indique les axes à suivre pour les recherches futures.

6.1. Résumé du document de thèse

La première partie de ce document présente le contexte, c'est-à-dire les systèmes complexes, les systèmes répartis, et la supervision de système. Par la suite, la partie sur les « problèmes industriels » présente deux systèmes complexes dans le domaine de fabrication de véhicules spatiaux (le système de la PDR et celui de DIS-RVM) qui illustrent les définitions et fournissent les motivations de la recherche actuelle. De plus, ces exemples servent de base de spécification du besoin pour un middleware de supervision et permettent d'en tirer des exigences. Ces exigences ont été fortement utilisées pour évaluer le concept en utilisant le prototype développé.

Le troisième chapitre présente un état de l'art dans le domaine des technologies et systèmes de supervision existants. Il identifie que des standards de supervision basés sur des protocoles (SNMP) et des interfaces/middleware (JMX) sont à la base de l'architecture des principaux systèmes de supervision (Unicenter TNG, OpenView, Tivoli etc.). Cette section fournit également une confrontation des systèmes existants vis-à-vis des exigences identifiées précédemment. Le manque d'adéquation constaté entre les solutions disponibles sur le marché et les besoins nécessite donc de poursuivre la recherche dans le domaine des middleware de supervision.

Le quatrième chapitre commence par une comparaison des technologies de middleware. Cette analyse laisse à penser qu'une solution basée sur les Web Services pourrait fournir des avantages liés à la fois à des systèmes de supervision basés sur des protocoles, ainsi qu'à ceux basés sur des middlewares. En conséquence, il a été décidé d'étudier l'applicabilité des Web Services en vue de la réalisation d'un middleware dédié à la supervision.

Dans la suite du chapitre, la présentation des Web Services montre la flexibilité de cette technologie et aide à la compréhension de la section sur l'architecture. De plus, elle décrit les concepts de base des technologies XML et fournit les jalons à venir concernant les activités en cours sur la prise en compte de la sécurité dans les Web Services.

Ensuite, le chapitre présent le point clé du document, c.-à-d. l'architecture générique d'un système de supervision basé sur les Web Services. Cette partie contient la description des composants de base, des opérations, le codage des informations de service et un exemple général d'encodage des données transportées. Cette architecture est l'innovation principale de ce travail et elle a été mise en application dans le projet GeneSyS pendant le prototypage.

Le quatrième chapitre se termine avec la description de la contribution de l'auteur au prototypage, qui a permis de vérifier le concept proposé dans un contexte industriel réel. Les systèmes de supervision développés pour les applications d'ingénierie collaborative (système PDR) et de simulations distribuées interactives (système DIS-RVM) utilisent l'approche proposée. En outre, les mécanismes simples mis en place pour la collaboration entre agent et la synthèse des informations démontrent la flexibilité du concept et sa complétude par rapport au besoin. L'utilisation d'un adaptateur en C++ résout efficacement les problèmes d'interopérabilité et fournit un moyen rapide pour développer de nouveaux agents.

Le cinquième chapitre présente une autre valeur ajoutée importante, c.-à-d. l'évaluation de l'applicabilité du concept. Les résultats principaux des essais de performance indiquent que les surcharges générales moyennes de la solution proposée (codage d'information) sont d'environ 38 kbps pour une paire de message requête/réponse. Bien que ce résultat soit habituel pour le monde XML, il indique clairement que la solution de surveillance n'est pas acceptable pour le système DIS-RVM au regard de ses spécifications (64kbps). Cependant, des tests plus récents sur des réseaux locaux et des réseaux hauts débits étendus, ont donnés des résultats acceptables, puisque pour le système DIS-RVM, la consommation moyenne de bande passante était d'environ 200kbps. D'ailleurs, l'approche appliquée utilise une surveillance périodique la plupart du temps, alors qu'une solution intelligente, utilisant des événements par exemple, diminuerait la charge de réseau de manière significative. Ainsi, cette approche basée sur des événements a été mise en application dans l'agent surveillant le MÄK RTI. Elle est donc disponible pour des recherches futures.

De plus, les essais de performance ont prouvé que dans certaines conditions (1 requête par 20ms), une solution basée sur les Web Services montre un temps de réponse (1 ms) comparable avec les middlewares basés sur des protocoles binaires. D'ailleurs, lors de tests de robustesse qui ont chargé le processeur à 100%, les agents se sont correctement comportés. Bien que le temps de réponse ait diminué de manière significative, aucune corruption ou perte de message n'ont été détectées. Par conséquent, ces tests prouvent la fiabilité et la robustesse du système dans le cas d'une utilisation intensive.

Les activités de validation par les utilisateurs réels ont démontré la viabilité du concept. Ces activités ont été organisées en différentes sessions mettant en jeu de vrais utilisateurs suivant des scénarios de validation permettant d'évaluer les fonctionnalités du middleware d'un point de vue utilisation. Durant cette phase de validation, le système a résisté aux pannes insérées par les utilisateurs, c.-à-d. le débranchement du réseau, la surcharge des processus du système d'exploitation, la terminaison anormale de certaines applications, etc.

Toutes ces actions ont permis de vérifier la conformité du middleware vis-à-vis des exigences décrites dans le chapitre d'introduction. De cette façon, les essais de performance laissent supposer que l'approche proposée est applicable pour un grand nombre de problèmes, où la flexibilité et l'interopérabilité de l'outil de supervision sont très importantes.

6.2. Impact Technologique

La section présente l'impact industriel de l'utilisation des Web Services dans le cadre de la supervision de systèmes repartis.

Récemment, plusieurs communautés ont fourni des spécifications de technologies de supervision basée sur les Web Services. Parmi celles-ci se trouvent les spécifications suivantes :

- OASIS MUWS - Management Using Web Services;
- OASIS-WSRF - WS-ResourceDescription, WS-Notification;
- WS-Management - WS-Eventing, WS-Enumeration de Microsoft;
- WSLA - Web Service Level Agreements.

Les architectures proposées dans ces spécifications ressemblent étonnamment au concept considéré dans ce document qui a été mise en application dans le projet GeneSyS. Au moment de la rédaction de ce document, les initiatives indiquées ci-dessus n'en sont qu'à la phase initiale d'implémentation. Il n'existe pas encore d'implémentations et d'applications réelles, tandis que dans le cadre de la validation du projet GeneSyS, des systèmes de supervision ont été créés pour les domaines suivants :

- Ingénierie Collaborative –Système PDR (EADS-ST);
- Simulation Distribuée Interactive – Système DIS-RVM (EADS-ST);
- Serveurs Web – Dictionnaire en ligne (MTA SZTAKI).

De plus, il est prévu de réutiliser les résultats de GeneSyS dans les domaines scientifiques et industriels notamment dans les domaines et projets suivants :

- Robotique Collaborative (University of Würzburg);
- Middleware de MMOG (Fraunhofer Gesellschaft);
- Plateforme mobile de jeux (University of Turin);
- « SMASH supercomputers based on FPGA devices »;
- Projets ASPIC et BROADWAN (EU IST FP6);
- Projet Tallisys CRUK.

Tout cela laisse supposer que le succès considérable du projet GeneSyS prouve explicitement l'applicabilité du concept proposé.

6.3. Sujets potentiels de recherches futures

Le travail décrit dans ce document ne répond pas à toutes les questions. Les sujets et préoccupations présentés ci-dessous montrent que la supervision de systèmes distribués demande encore de nombreuses recherches, tant académiques qu'industrielles :

- Une architecture commune pour les adaptateurs du middleware pourrait aider à développer des modèles génériques de programmation des agents de supervision;
- Une analyse en vue d'améliorer le codage des données échangées aiderait à optimiser le trafic de réseau ;
- Une chorégraphie d'agents de supervision élaborerait des possibilités de services d'auto-découverte, d'auto-configuration et améliorerait la collaboration entre les agents ;

- Les recherches dans le domaine des réseaux pair-à-pair d'agents de supervision semblent être une suite logique d'une chorégraphie d'agents;
- Des recherches ultérieures dans le domaine de l'intelligence de la supervision, de la prévision des réactions du système en fonction de comportements permettraient de concevoir un système de réparation automatique ;
- La collaboration avec les groupes de standardisation comme WBEM (CIM) et WSDM (OASIS) permettrait de normaliser le concept.

6.4. Postface

Selon SUN Microsystems, l'ordinateur est un réseau.

Selon Intel, l'évolution des systèmes répartis passe par les réseaux Pair-à-Pair.

Selon Microsoft, les Web Services sont une panacée.

En tout cas, la complexité des problèmes induits par la distribution des systèmes affirme le besoin de systèmes de supervision.

CONTENTS:

TABLE OF FIGURES

LIST OF ACRONYMS:

API	Application Programming Interface
ATV	Automated Transfer Vehicle
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
DBMS	Database Management System

DCE	Distributed Computing Environment
DCOM	Distributed COM
DIS	Distributed Interactive Simulation
DIS-RVM	Distributed Interactive Simulation of the Rendez-Vous Mission
DOM	Document Object Model
EADS-ST	EADS SPACE Transportation
EJB	Enterprise Java Beans
ESA	European Space Agency
GeneSyS	Generic System Supervision
GIOP	General Inter-ORB Protocol
GUI	Graphical User Interface
HLA	High Level Architecture
HMI	Human Machine Interface
HTTP	Hyper Text Transfer Protocol
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
ISS	International Space Station
JMF	Java Media Framework
JMS	Java Messaging Service
JNI	Java Native Invocation
JMX	Java Management Extension
LAN	Local Area Network
MOM	Message Oriented Middleware
MTS	Microsoft Transaction Server
NTP	Network Time Protocol
OMG	Object Management Group
OSF	Open Software Foundation
OSI	Open System Interconnect
ORB	Object Request Broker
PDR	Preliminary Design Review
QoS	Quality of Service
RID	Review Item Discrepancy
RMI	Remote Method Invocation
RPC	Remote Procedure Call

RTI	Run-Time Infrastructure
SAX	Simple API for XML
SCS	Supervision and Control System
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
UDDI	Universal Description, Discovery and Integration Protocol
UML	Unified Modelling Language
URL	Universal Resource Locator
WAN	Wide Area Network
WSDL	Web Services Description Language
WSFL	Web Services Flow Language
XML	eXtended Mark-up Language

Chapter 1. INTRODUCTION

1.1 PROBLEM STATEMENT

During the past years the Information Technology domain showed a continuous trend from a single computer towards complex networks and from monolithic systems to multi-modular distributed ones. The distributed systems require an appropriate supervision solution both on network and application levels. For this purpose, several supervision distributed management standards and frameworks are currently available, such as SNMP [9], JMX [10], Tivoli (from IBM) [11], OpenView (from HP) [12] and NAGIOS [13]. They are aimed at different aspects of monitoring starting from operating systems through network and up to some commercial standard applications.

However most of them have several common constraints, some of them are mentioned in the list below:

- *Interoperability issues:* Components written in different languages using different toolkits, which are supposed to use the same architecture specification, may not be capable to co-operate on a full scale.
- *Component portability:* Often components are built to work only under their native operating systems like MS-Windows or Linux. They are very sensible to transport mechanism and more generally speaking to low-level communication protocols.
- *Development/deployment complexity:* Many commercial applications have proprietary APIs what makes it difficult to create new agents and plug them to existing supervision system.
- *Non-flexible architecture:* When agent and visualisation tools are released in the same component, upgrades of console impact agent functionality and visa versa.
- *Dedication to a particular monitoring layer, lack of comprehensive solutions:* There exist many different solutions for monitoring of different individual aspects. Meanwhile an integral supervision framework would allow to evaluate simultaneously an overall system status. For instance, the health of an application heavily depends on the health of an operating system. As monitoring of a single application would not give a correct overview of a situation, it is necessary to integrate the information from all the system components involved.

The above-mentioned constraints complicate integration between third-party monitoring tools. This integration is essential to ensure system control on all the levels. Besides, proprietary solutions slow down the pace of development of the whole domain.

Therefore, the research objective is to investigate a possibility of an **open, generic modular and comprehensive supervision middleware concept**, which would solve or alleviate the above-mentioned limitations.

1.2 CONTRIBUTIONS

The current PhD research was performed in the frame of the GeneSyS project at Laboratoire d'Informatique de l'Universite Paris 6 and at EADS SPACE Transportation.

The author's contributions can be briefly summarized as follows:

- Investigation of two distributed systems in the space industry (a Collaborative Engineering Application and a Distributed Interactive Simulation) in order to determine common supervision requirements;
- Analysis of the state of the art in the distributed supervision domain in order to study their applicability for the industrial systems;
- Analysis of the middleware technologies to bound possible solutions;
- Design of a supervision framework based on Web Services;
- Design and implementation of a prototype supervision solution for the Collaborative Engineering Application and for the Distributed Interactive Simulation;
- Demonstration of the concept applicability by means of two validation scenarios;
- Evaluation of performance characteristics of the proposed solution.

1.3 METHODOLOGY AND OUTLINE

The contributions listed in the previous section are described by the current document, which is logically structured as it is outlined below:

- **Chapter 2**: The section **Problem Context** intends to give the main **Definitions** considered in the document: Complex Systems, Distributed Systems, Supervision Systems. These definitions are illustrated with real life examples: Preliminary Design Review application (Collaborative Engineering) and Distributed Training Scenario (Interactive Simulation of the Rendez-Vous Mission). The goal of these subsections is to provide representative requirements, i.e. evaluation criteria for the resulting generic supervision middleware. In this way, all the analysis in the document is conducted in the light of these criteria;
- **Chapter 3**: The **State of the Art** section provides a pass through existing supervision technologies, frameworks and research projects. The chapter ends with an analysis of the existing solutions and their applicability to the above-mentioned industrial problems according to outlined requirements;
- **Chapter 4**: The **Concept and Implementation** sections focus on the innovation. The chapter starts with a hypothesis, that the Web Services could be a good choice to build the required architecture on top of it. The analysis of the available

middleware technologies intends to justify this choice. Afterwards the Web Services technology section introduces the main basics of the technology for better understanding of the proposed approach. The **Proposed Architecture** section describes an approach that maps a supervision system (see. **Definitions**) to Web Services. In the end, the **Prototype** section elaborates approaches and solutions for supervision of the required industrial systems;

- **Chapter 5**: Finally, the **Evaluation** section is intended to discuss solution limitations. First, it provides results of performance tests. Then, it describes the validation process, which involved real users. In addition, this chapter argues about compliance of the proposed concept with the problem criteria.

Chapter 2. PROBLEM CONTEXT

2.1 DEFINITIONS

The purpose of this section is to define the main terms considered in this document: Complex Systems, Distributed Systems, Supervision Systems, Supervision Middleware. The second part illustrates these definitions by means of two industrial systems. These systems serve as an important testbed for evaluation of the resulting concept. In order to define evaluation criteria, the last part of this section summarises common requirements for a supervision middleware.

2.1.1 COMPLEX AND DISTRIBUTED SYSTEMS

To proceed further with the problem definition, the “Complex System” term should be clarified.

According to the WordIQ dictionary [2], “complex systems” have a number of properties, some of which are listed below. It is also often used as a broad term addressing a research approach, which includes ideas and techniques from chaos theory, artificial life, evolutionary computation and genetic algorithms.

The representative properties of complex systems include:

- **Emergence:** What distinguishes a complex system from a merely complicated one is that some behaviours and patterns emerge in complex systems as a result of the patterns of relationship between the elements. Emergence is considered as the key property of complex systems.
- **Relationships are short-range:** Typically, the relationships between elements in a complex system are short-range, that is information is normally received from near neighbours. The richness of the connections means that communications will pass across the system but will probably be modified on the way.
- **Relationships are non-linear:** There are rarely simple cause and effect relationships between elements. A small stimulus may cause a large effect or no effect at all.
- **Relationships contain feedback loops:** Both negative (damping) and positive (amplifying) feedback are key ingredients of complex systems. The effects of an

agent's actions are fed back to the agent and this, in turn, affects the way the agent behaves in the future. This set of constantly adapting nonlinear relationships lies at the heart of what makes a complex system special.

- **Openness:** Complex systems are open systems - that is, energy and information are constantly being imported and exported across system boundaries.
- **Complex systems have a history:** The history of a complex system is important and cannot be ignored. Even a small change in circumstances can lead to large deviations in the future.
- **Complex systems are nested:** Another key aspect of complex adaptive systems is a hierarchy of the components of the system. For example, an economy is made up of organisations, which are made up of people, who are systems of organs controlled by their nervous and endocrine systems, which are made up of cells - all of which, at each level in the hierarchy, are complex adaptive systems.
- **Absence of boundaries:** It is usually difficult to determine the boundaries of a complex system.

In the frame of this work, a “Complex System” is considered in the context of distributed systems. A distributed system [3] is a collection of autonomous computers linked by a network and equipped with distributed system software. It is the opposite of a centralized system, which consists of a single computer with one or multiple powerful CPUs processing all incoming requests. The distributed system software enables the comprising computers to coordinate their activities and to share system resources. Well-developed distributed system software provides the illusion of a single and integrated environment although it is actually implemented by multiple computers in different locations. In other words, the software gives a distribution transparency to the systems.

The systems of this kind comprise the following properties:

- Components are **distributed**: for example, systems that implement client/server or peer-to-peer architecture and physically hosted on different computers;
- **Heterogeneous** components are involved: for example, often components are coded on different languages and run under different operating systems;
- System involves several **layers**: for example, a system that is composed of operating system, network, middleware and application layers.
- The **number of components** is high.
- The **number of** components **relations** is high.

A distributed system generally has several important characteristics [3], obtained in the result of a careful design and implementation:

- **Resource Sharing:** Resources provided by a computer, which is a member of a distributed system, can be shared by clients and other members of the system via a network.
- **Openness:** Openness in distributed systems is the characteristic that determines whether the system is extendible in various ways. This characteristic is measured mainly by the degree to which new resource sharing services can be incorporated without disruption or duplication of existing services. The opposite of an open distributed system is a closed distributed system. The set of features and facilities

provided by a closed distributed system stay static overtime. New features and facilities cannot be added into the system. This prevents the system from providing any other new resources other than those, which have already been made available. Systems with proprietary interfaces are often referred to as closed.

- **Concurrency:** Concurrency is the ability to process multiple tasks at the same time. A distributed system comprises multiple computers, each having one or multiple processors. The existence of multiple computation resources can be exploited to perform multiple tasks at the same time. This ability is crucial to improve the overall performance of the distributed system. For example, a mainframe must handle requests from multiple users, with each user sending multiple requests at the same time. Without concurrency, performance would suffer, since each request must be processed sequentially.
- **Scalability:** Scalability in distributed systems is the characteristic where a system and application software does not need to change in case the scale of the system increases. Scalability is important since the amount of requests processed by a distributed system tends to grow, rather than decrease.
- **Fault Tolerance:** Fault Tolerance in distributed systems is a characteristic where a distributed system provides an appropriate handling of errors that occurred in the system.
- **Transparency:** Transparency means that the separation of components in a distributed system does not affect the way users and application programmers view the system, which means that they perceive the system as a whole rather than a collection of independent components. In other words, the separation is hidden.

The trend of distributed systems is motivated due to several potential benefits:

- **Shareability**: Shareability in distributed systems is the ability that allows the comprising systems to use each other’s resources.
- **Expandability**: Expandability of a distributed system is the ability that permits new systems to be added as members of the overall system.
- **Local Autonomy:** A distributed system is responsible to manage its resources; therefore, it gives its components a local autonomy of their resources.
- **Improved Performance:** As the number of clients accessing a resource increases, the response time starts to degrade. The conventional ways of maintaining the response time, for example, upgrading the host machine, can be used to offset this effect. The separate nature of distributed systems can improve under certain conditions the situation by distributing and balancing the computation load.
- **Improved Reliability and Availability:** Distributing the computation load through several stations makes the service still available in case when one of the stations is down.
- **Potential Cost Reductions:** One of the most frequent examples of cost reduction is a situation when a computation server is shared between several clients, contrarily to monopoly occupation of the computation resource.

<u>Architecture Basics:</u>

Two following simplified models illustrate the above-mentioned properties. The first of them, a generalised component model (Figure 1) is purposed to show a distribution aspect and simple binary relations.

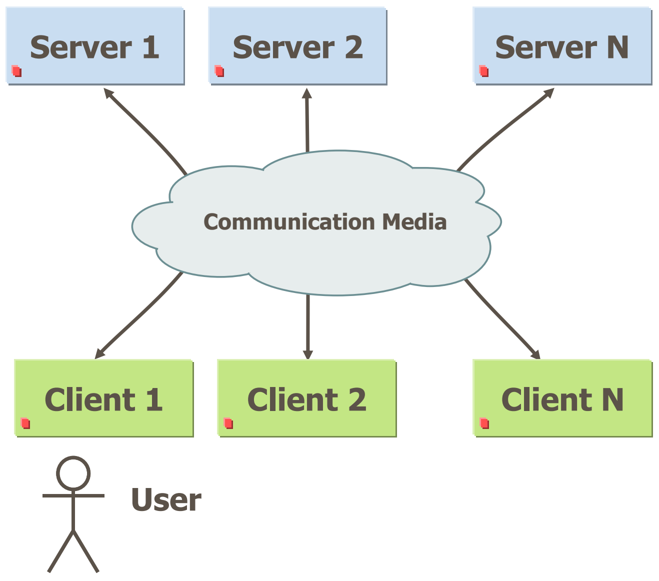


Figure 1. General Component Model of a Complex System

The server component represents a module that provides some services, while the client component is a service consumer. Communication media provides a means to interconnect clients and servers. There can be different means of communication like network protocols (TCP, UDP, HTTP, etc), inter process communication means (shared memory, pipes, etc) or middleware (RPC, CORBA, JAVA RMI, Message Oriented Middleware, etc).

Figure 1 is very schematic. In reality, modern distributed systems usually implement more complex relations, like one-to-many and many-to-many. In this case, a component may represent a client for one group of components and be a server for another one, which is often called the peer-to-peer communication model.

Regarding the PDR scenario, which will be described later in this section, there exist several server components, like Tomcat Server, visio conference server (MCU), Oracle database server, etc, as well as several client components (EDB client, visio conference client - MCU, etc.). As for the Communication media, it is secured by the network protocols.

The next model, which is a generalised component deployment model (Figure 2), shows a component execution platform, which is essential for the system operation.

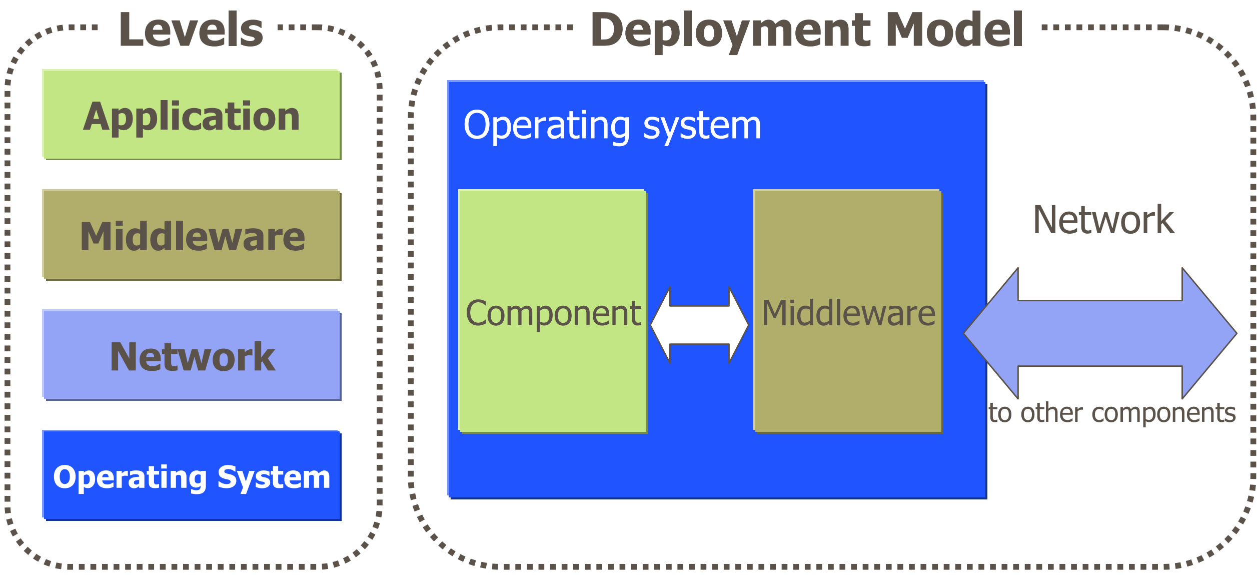


Figure 2. General Component Deployment Model

The components are classified hierarchically according to several levels:

- **Operating system** managing component execution and inter-process communication;
- **Network** securing communication with other remote components;
- **Middleware** hiding details of system distribution;
- **Application** dealing with the application logic.

The above levels will also be used later to classify the supervision parameters and agents.

2.1.2 Distributed Supervision Systems and Supervision Middleware

Beside advantages mentioned in the previous section, a distributed system has the following disadvantages [3]:

- **Network Reliance:** Because all computers in a distributed system rely on a network to communicate with each other, problems on the network could disrupt activities in the system as a whole. This is true especially for physical problems such as broken network cables, routers, bridges, etc. The cost of setting up and maintaining the network could eventually outweigh any potential cost savings offered by distributed systems.
- **Complexities:** Administrators must be able to deal with different errors that could occur from all components of a distributed system and their deployment platforms that make up the distributed system. It must also be capable to manipulate resources of computers with a wide range of heterogeneities.
- **Security:** A distributed system allows its computers to collaborate and share resources more easily. However, this convenience of access could be a problem if no proper security mechanisms are put in place. Private resources would be exposed to a wider range of potential hackers, with unauthorized accesses launched

from any computers connected to the system. In fact, a centralized system is usually more secure than a distributed system.

In order to alleviate these issues, it is a common practise to use distributed supervision systems.

In general, a **supervision system** is defined as an application providing following functionality:

- Acquisition of monitoring data;
- Representation of management data on a GUI console;
- Support of supervised system control (manual, semi or fully automated).

A **distributed supervision system** acts via components called **supervision agents** sharing properties of general use software agents. Basically, a software agent is an autonomous entity with its own ontology and schedule. Each agent possesses the ability to act autonomously; this is an important distinction because a simple act of obedience to a command does not qualify an entity as an agent. An agent may interact or negotiate with its environment and/or with other agents. It may make decisions, such as whether to trust or to cooperate with others. In addition to these features, supervision agents implement particular functionality depending on their purposes.

Figure 3 depicts a general architecture of a distributed supervision system.

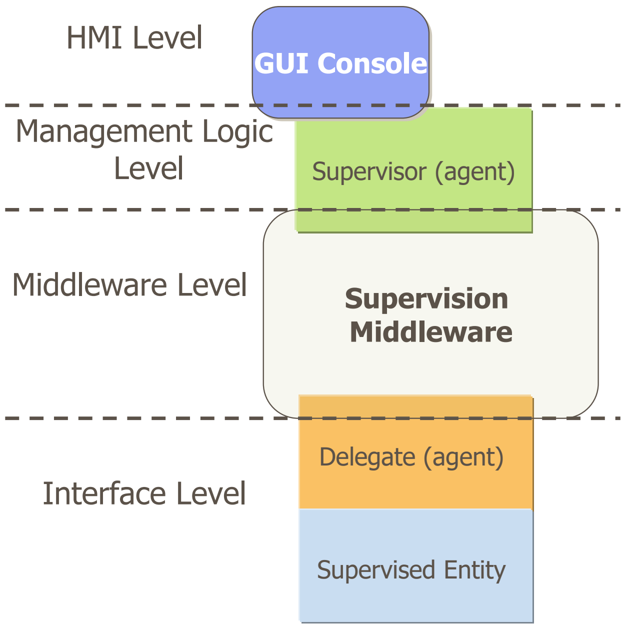


Figure 3. General Architecture of a Distributed Supervision System

A distributed supervision system generally has an interface to a **supervised entity** (operating system, database server, etc). Via this interface supervision, an agent registers parameters fluctuations, incoming events, executes commands and coopers with a **GUI console**. The GUI console can be integrated or remote; in this case, specific communication mechanisms are applied. They can be inter-process communication means (files, shared memory, pipes, etc), network protocols (TCP, UDP, HTTP, SNMP, etc.), common use middleware (JMX, CORBA, FIPA, etc.), or others. These mechanisms are actually of major interest for this work.

When distributed, supervision agents are classified as **Delegates** and **Supervisors**. Delegates implement an interface to a supervised entity and act on behalf of Supervisors, as a representative of the system administrator. In its turn, Supervisors operate with monitoring data and control functionality implementing management logic, such as information evaluation and summarising, behaviour patterns recognition, decision making and automated entity control.

Graphic user interface console is purposed for a human machine (HMI) communication. It represents the monitoring synoptics, global system status evaluation or summary. In addition, it forwards administrator commands to the supervision system.

As for the **supervision middleware**, similarly to the general use middleware, its goal is to centrally provide high-level abstractions and services to the agents, to ease agent programming, integration, and system management tasks. In this sense, middleware moves beyond transaction management to encompass database management systems, web servers, application servers, content management systems, and similar tools that support the agent development and data delivery process. Hence, the main purpose of the supervision middleware is to provide developers with the following facilities:

- abstraction to the agents;
- a data (including monitoring information and commands) delivery mechanism;
- distribution services, for example, directory and repository services;
- agent development patterns.

The state of the art survey in the next sections supplies outlines for a better understanding of the modern supervision middleware technologies and frameworks.

2.2 INDUSTRIAL PROBLEMS

This section is devoted to shortly describe two industrial examples of complex systems, which require the supervision. Both of them are closely related to EADS SPACE Corporation business on the design of the Automated Transfer Vehicle (ATV) spacecraft. Consequently, the first subsection shortly describes the ATV mission. While the second subsection, called Preliminary Design Review (PDR), presents one of the stages of ATV design and a support system, which is to be supervised. The last problem is referred to as the Distributed Training Scenario concerns distributed interactive simulation of the rendez-vous mission (DIS-RVM) and illustrates the need of supervision in the domain of real-time simulators. The proposed concept will be evaluated on these two systems. Hence, more detailed information can be found in section 0.

2.2.1 AUTOMATED TRANSFER VEHICLE

The ATV (Automated Transfer Vehicle) is an unmanned spacecraft developed by ESA and is to be launched by Ariane 5. It has three main mission goals: to deliver freight (scientific payloads, propellant, water, etc.) to the International Space Station (ISS), to re-boost the station and to take away any waste from the ISS. Figure 4 illustrates schematically a typical ATV mission.

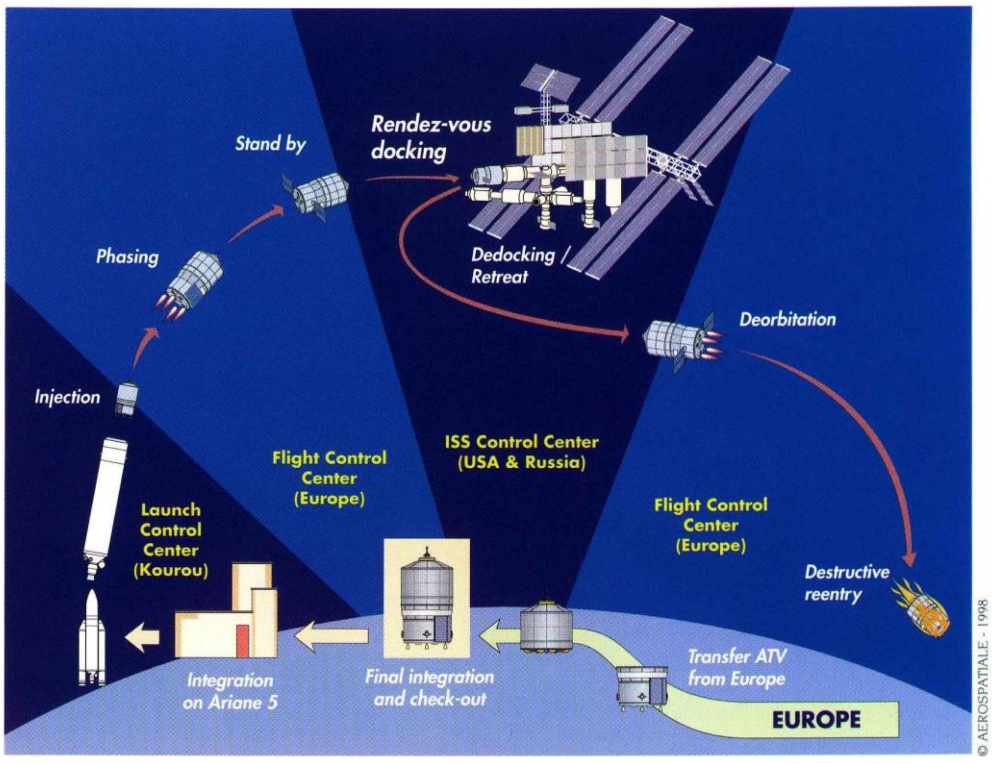


Figure 4. ATV Mission

2.2.2 Preliminary Design Review

The Preliminary Design Review (PDR) [4] is a major milestone in the development cycle of complex systems like spacecraft in EADS-ST scope. The GeneSyS project took the Automated Transfer Vehicle (ATV) PDR as a validation scenario (the real PDR has been performed in a traditional way).

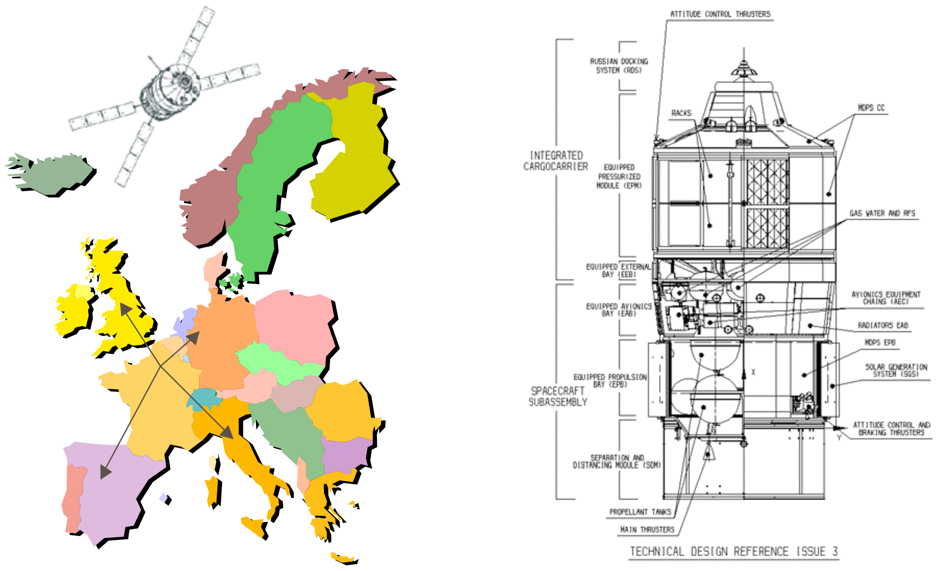

ATTITUDE CONTROL THRUSTERS
RUSSIAN DOCKING SYSTEM (RDS)
RACKS
MDPS CC
INTEGRATED CARGOCARRIER
EQUIPPED PRESSURIZED MODULE (EPM)
GAS WATER AND RFS
EQUIPPED EXTERNAL BAY (EEB)
EQUIPPED AVIONICS BAY (EAB)
AVIONICS EQUIPMENT CHAINS (AEC)
RADIATORS EAB
MDPS EPB
EQUIPPED PROPULSION BAY (EPB)
SPACECRAFT SUBASSEMBLY
SOLAR GENERATION SYSTEM (SGS)
ATTITUDE CONTROL AND BRAKING THRUSTERS
SEPARATION AND DISTANCING MODULE (SDM)
PROPELLANT TANKS
MAIN THRUSTERS
TECHNICAL DESIGN REFERENCE ISSUE 3

Figure 5. PDR process for ATV design

EADS-ST as a prime contractor on behalf of ESA (European Space Agency) leads this programme. During the Program life-cycle several reviews have been planed. The PDR is one of the first reviews among them [4]. During the PDR, engineers from different countries collaborate on ATV design documentation, create Review Items Discrepancies (RID), meet on-line to discuss RIDs and possibly release Change Proposals. To support these activities a Distributed Engineering system, called PDR application, is applied (Figure 6).

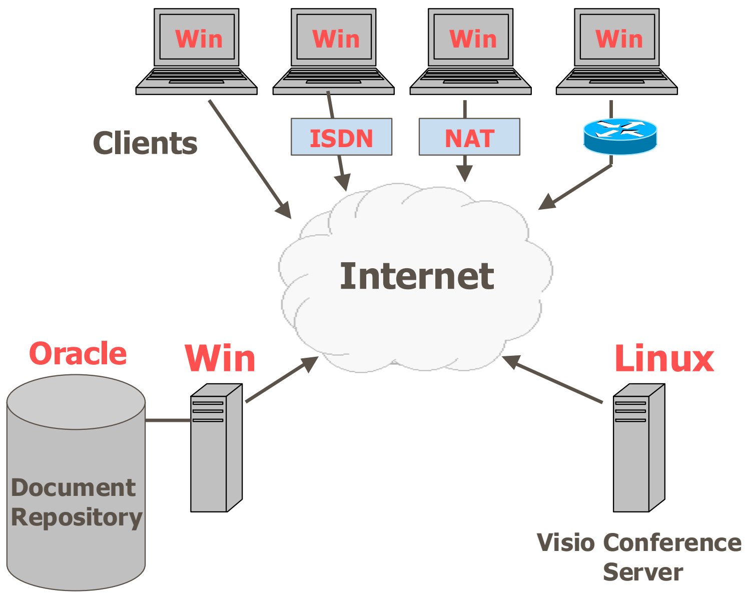


Figure 6. PDR Application

This system consists of a document repository, called Engineering Database (EDB) server, and Visio conference server, which routes H.323 and T.120 protocol streams and manages sessions (on-line meetings). The EDB server contains thousands of documents. Both servers support up to one hundred users simultaneously. Review meetings involve up to fifty engineers.

Although the servers are hosted on Linux, client applications, documentation front-end and video chat are run under Windows. The network is heterogeneous, since reviewers are located in different countries and they use various server access means, including ISDN and Ethernet networks. Moreover, some client hosts are hidden by firewalls and network address translation. As far as the application layer is concerned, the EDB was developed on the basis of Java e-business solutions, while the conference server was developed on C++ and uses PHP based application for session management. These constraints determine the supervision solution.

To efficiently manage the PDR process, a specific supervision solution is required. The supervision is needed to administrate operating systems, network and applications involved. There exist numerous heterogeneous ways to monitor and control all aspects of the system, such as specific APIs and plug-ins for Database Management System, Network Management Protocols, Toolkits for system monitoring. Therefore, the generic way of the information gathering, synthesising, and representation is actually required. This system should give us a global view of the situation to simplify the management and maintenance.

2.2.3 DISTRIBUTED TRAINING SCENARIO

This scenario deals with docking phase of the ATV mission. The selected spacecraft rendezvous and docking scenario will focus on the final translation of ATV to ISS, which is one of the most critical phases of the whole ATV mission. This phase starts at a waiting point S3, at approximately 250 meters from the station. From this point, the ATV spacecraft follows its docking procedures. The trajectory is represented below in Figure 7.

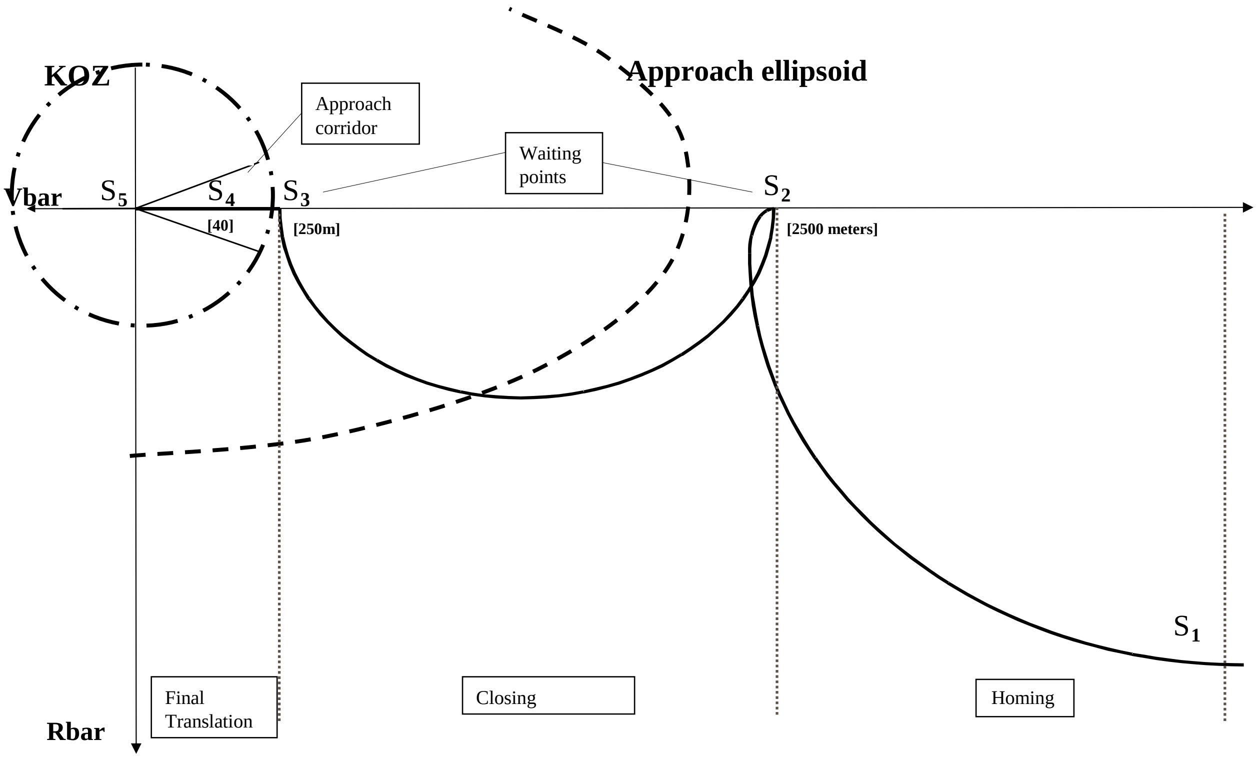


Figure 7. Overall view of the ATV flight trajectory near ISS

Interoperability between ATV simulation and training facilities could greatly facilitate test and training phases required for such complex operations. The distributed simulation technology is considered as a solution for effective exchange of data between simulators and simulations.

In the space domain, it is essential to prepare in advance the astronauts and the ground team to the worst cases they might face during a mission, as statistics show that only 35% of contingency situations are really predictable. Therefore, it is mandatory to train them intensively on simulators, which are the most representative.

With regard to the training of geographically distributed groups of operators, it is of critical importance to have capabilities to effectively supervise various components of such a complex distributed training system.

DIS-RVM software is the main component of this training system. It represents distributed interactive simulation of the ATV-ISS rendez-vous mission based on HLA (High Level Architecture) middleware [5].

The following HLA federation of the DIS-RVM software is used (Figure 8):

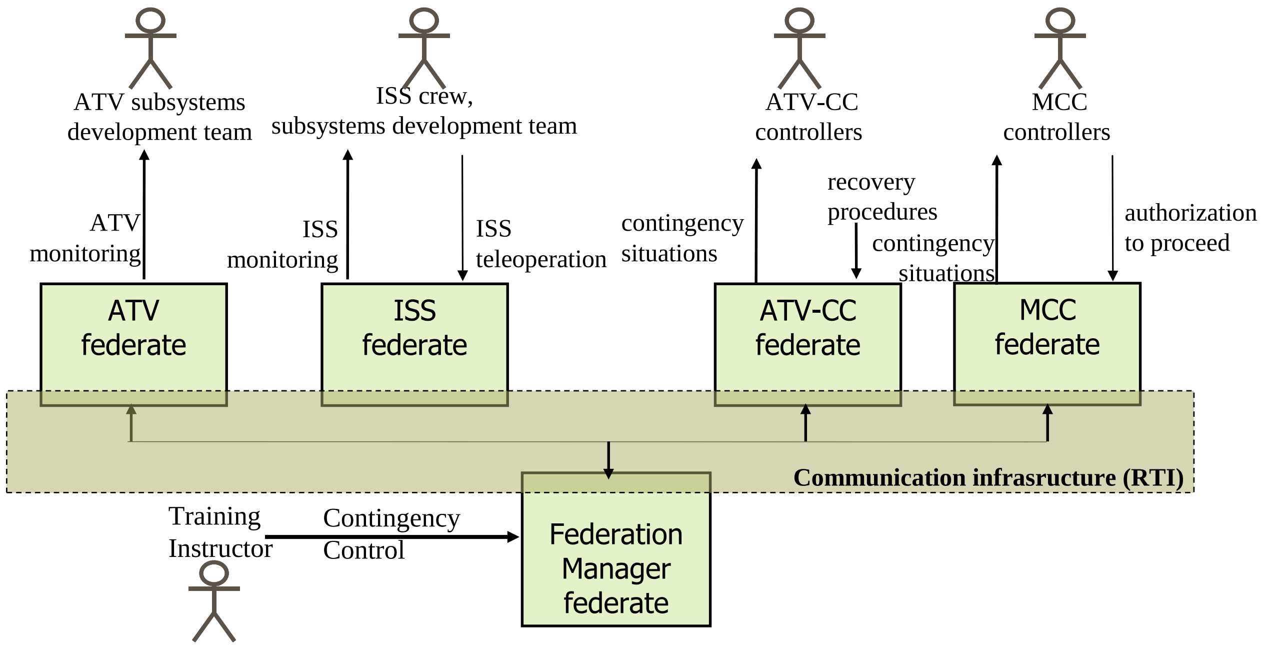


Figure 8. Architecture of the DIS-RVM Federation

All federates are independent Linux applications that can run within LAN (a local area network) or WAN (a wide area network), which is due to provide the following minimum quality of service:

- Latency of 300 milliseconds
- Bandwidth of 64 kbps
- Routing of multicast

DIS-RVM federates are “soft real time” applications with a maximum jitter of model scheduling of less than 10 milliseconds on single CPU Linux hosts. The federates may run in faster-than-real-time mode with a time factor of up to four.

To provide consistency of simulation data within the distributed simulator and the level of fidelity necessary for training purposes, federates use uniform time reference obtained by means of NTP (Network Time Protocol). Synchronisation of system clocks on simulation hosts is necessary to run simulation (training) sessions.

ATV federate simulates the orbital flight of ATV and functioning of major ATV onboard systems during the Rendez-Vous & Docking (RVD). In addition, it simulates essential functionality such as an intervention of a remote operator into the control loop and introduction of failures. The main functionality of the ATV federate is as follows:

- simulation of orbital flight of the ATV;
- monitoring of the progress of the RVD process;
- possibility to control the ATV motion during the simulation.

ISS federate simulates the orbital flight of ISS and functioning of the ISS attitude control system. Regarding centre-of-mass control, the ISS is simulated as a passively moving space vehicle. The main functionality of the ISS model is as follows:

- simulation of the orbital flight of a space station;
- monitoring of the progress of the RVD process;

- possibility to remotely control the ATV during the simulation.

ATV-CC federate represents the most important functionality of the ATV Control Centre such as:

- monitoring the ATV parameters and variables;
- managing the RVD sequence in case of contingency situations.
- modifying the ATV flight plan parameters;
- controlling the ATV motion remotely by a ground operator.

MCC federate represents the most important functionality of the Mission Control Centre such as:

- monitoring the ISS parameters and variables;
- controlling the RVD sequence in case of contingency situations due to ATV: GO/NO-GO, etc;
- managing the ATV flight plan;
- managing the authority transfer process.

Federation Manager federate manages the distributed simulator and introduces contingencies independently from any federate. Its functionality provides for the following:

- setting of exercise-specific input data;
- checking the process of loading and initialising of models;
- managing time factor during the exercise;
- issuing commands such as START, STOP, PAUSE;
- introducing off-nominal situations.

DIS-RVM can interoperate with number of image generation systems for realistic visualization of ATV rendez-vous. Example of such a generated scene is given in Figure 9.

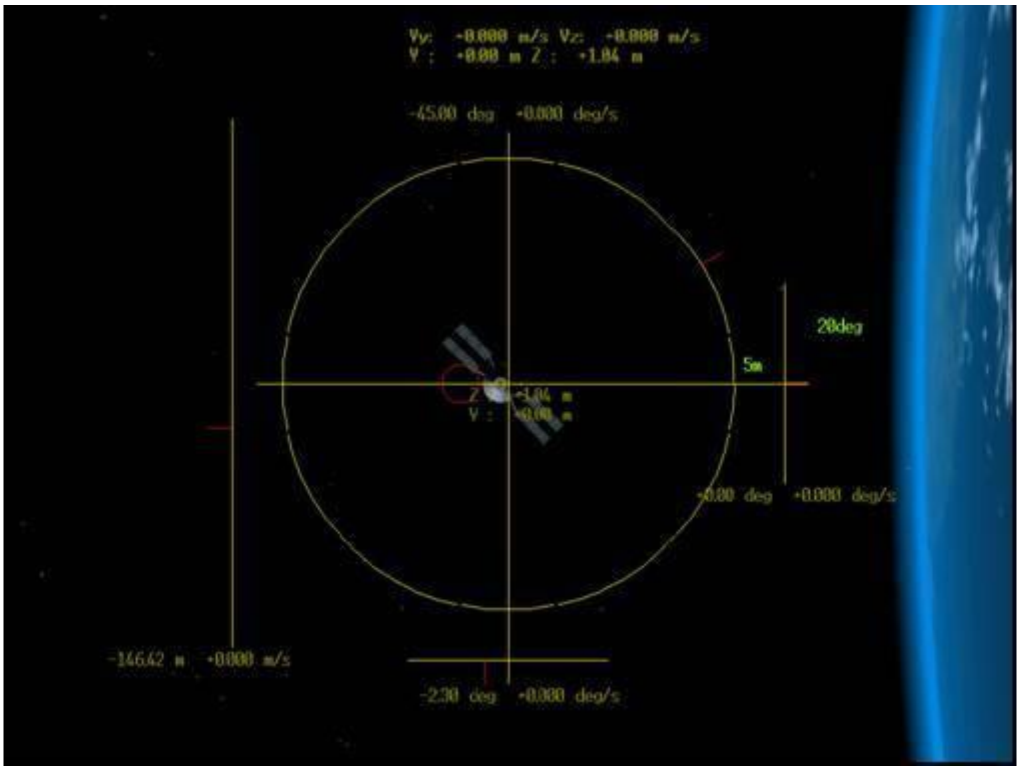


Figure 9. Visual representation of DIS-RVM simulation

To effectively monitor and control this application, the supervision system should provide information about parameters from all the system levels:

- **Application:** Simulation sessions, Simulation events, Federate state, Federate events, Federate logs, Federate Activities on the HLA API and network level, RTI (Run-Time Infrastructure) availability, RTI state, RTI events, RTI logs, etc;
- **Network:** Network resources load, Network connectivity, Network bandwidth, Network interface load, System logs, Interfaces' Logs, Firewall Logs, etc;
- **Operating system:** System resources load, Disk load, CPU load, Memory load.

The supervision middleware should allow collecting the monitoring data from heterogeneous sources. Moreover, as in the previous case, a global synthetic view of the system status is the main goal for supervision.

2.3 REQUIREMENTS FOR SUPERVISION MIDDLEWARE

In addition to distribution management, the following requirements have been identified on the basis of the analysis of the above-mentioned industrial problems:

- **Interoperability:** The supervision middleware should be capable to interconnect heterogeneous agents. That includes agents, which have been coded in different programming languages (for the above-described problems, the programming languages include Java, C++, PHP), or/and agents which run under different operating systems (like Windows and Linux).

- **Portability:** the middleware components should be portable to run under different operating systems (like Windows and Linux) and to be capable to operate over different network protocols (like UDP, TCP).
- **Flexibility:** the middleware should be capable to transport all types of data including complex (arrays, lists, vectors, hash tables) and compound data.
- **Support of intelligence:** the middleware should be capable to transport additional custom information dedicated to intelligent supervision.
- **Reliability and security:** the middleware operation should be reliable. It should provide authentication, authorisation and data encryption functionality.
- **Ease of agent development:** The middleware should provide agent development patterns, which should simplify integration with the middleware.
- **Ease of middleware deployment:** The middleware components should be easy to install and configure.

To evaluate compliance to these requirements, it is a common practise to develop prototypes dealing with representative systems. In the frame of this work, the requirements are verified using prototypes for supervision of the PDR and DIS-RVM systems (see Prototype section); therefore, the compliance is validated based on the following requirements:

Requirement:	PDR system	DIS-RVM system
Interoperability	Java, C++, C#, PHP agents Windows, Linux platforms	
Portability	Agents should be able to develop using Java, C++, C#, PHP and to run under Windows, Linux platforms	
Flexibility	Support of complex and compound types for supervision data	
Intelligence	-	Global view: information summarisation
Reliability and security	Reliability of design is verified playing the validation scenarios. Authentication, authorisation and data encoding support.	
Ease of development	Design patterns for Java, C++, C#, PHP	
Ease of deployment	Regarding PDR execution platform	Regarding DIS-RVM execution platform

The further information about the validation process is given in the Evaluation chapter.

The following chapter is intended to overview and to evaluate current supervision middleware technologies in accordance with the above-mentioned requirements.

Chapter 3. STATE OF THE ART

This section describes the current situation in the domain of supervision middleware, their architecture and application. It is followed by the analysis of advantages and limitations of the existing supervisory solutions in regard to the industrial problems. This analysis is intended to design a more appropriate solution. In addition, the analysis of the underlying middleware technologies is provided in the next chapter.

3.1 NETWORK MANAGEMENT PROTOCOLS

Network management technologies have been developed during the whole history of networks. The best known of them are:

- *High-level Entity Management System (HEMS)*;
- *Simple Gateway Monitoring Protocol (SGMP)* ;
- *Common Management Information Protocol (CMIP)*.

These technologies highly contributed to the development of the ICMP and SNMP protocols described hereafter.

3.1.1 INTERNET CONTROL MESSAGE PROTOCOL (ICMP)

ICMP is an error-reporting system. It is an integral part of IP and must be included in every IP implementation. The complete specification of ICMP is ***RFC792*** [8]. ICMP is responsible for reporting errors and messages regarding the delivery of IP datagrams. It can also send "source quench" and other self-tuning signals during the transfer of data between two machines without the intervention of the user. These signals are designed to fine-tune and optimise the transfer of data automatically. ICMP is the protocol that warns when a destination host is unreachable, or how long it took to get to a destination host.

Some of ICMP's functions serve to:

- Announce network errors, such as a host or entire portion of the network being unreachable, due to some type of failure. A TCP or UDP packet directed at a port number with no receiver attached is also reported via ICMP.
- Announce network congestion. When a router begins buffering too many packets, due to inability to transmit them as fast as they are being received, it will generate ICMP Source Quench messages. Directed at the sender, these messages should cause the rate

of packet transmission to be slowed. Of course, generating too many Source Quench messages would cause even more network congestion, so they are used sparingly.

- Assist Troubleshooting. ICMP supports an Echo function, which just sends a packet on a round-trip between two hosts. Ping, a common network management tool, is based on this feature. Ping will transmit a series of packets, measuring average round-trip times and computing loss percentages.
- Announce Timeouts. If an IP packet's TTL (Time To Live) field drops to zero, the router discarding the packet will often generate an ICMP packet announcing this fact. TraceRoute is a tool, which maps network routes by sending packets with small TTL values and watching the ICMP timeout announcements.

The ICMP protocol uses IP addressing because it is a protocol encapsulated within an IP datagram. Figure 10 illustrates the fields of an ICMP message.

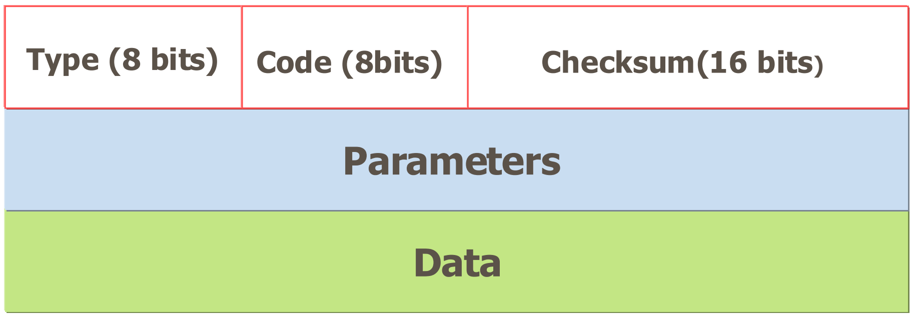


Figure 10. Layout of an ICMP message

The first field is the ICMP message type, which can be classified as either a query or an error. The code field further defines the type of query or message. The checksum field is the 16-bit one's complement sum of the ICMP header. Finally, the ICMP contents depend on the ICMP type and code.

ICMP messages can be broken down into two basic categories: the reporting of errors and the sending of queries. Error messages include the following:

- Destination unreachable
- Redirect
- Source quench
- Time exceeded

ICMP also includes general message queries. The two most commonly used are the following:

- Echo request
- Echo reply

The most familiar tool for verifying that an IP address on a network actually exists is PING utility. The PING utility reports the existence of the IP address and how long it took to get there.

3.1.2 Simple Network Management Protocol (SNMP)

Simple Network Management Protocol (SNMP) [9] 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.

Next section will present basics of the SNMP concept.

<u>Architecture:</u>

The SNMP is application level protocol on top of UDP. The SNMP managed network consists of tree major components (Figure 11): 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 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.

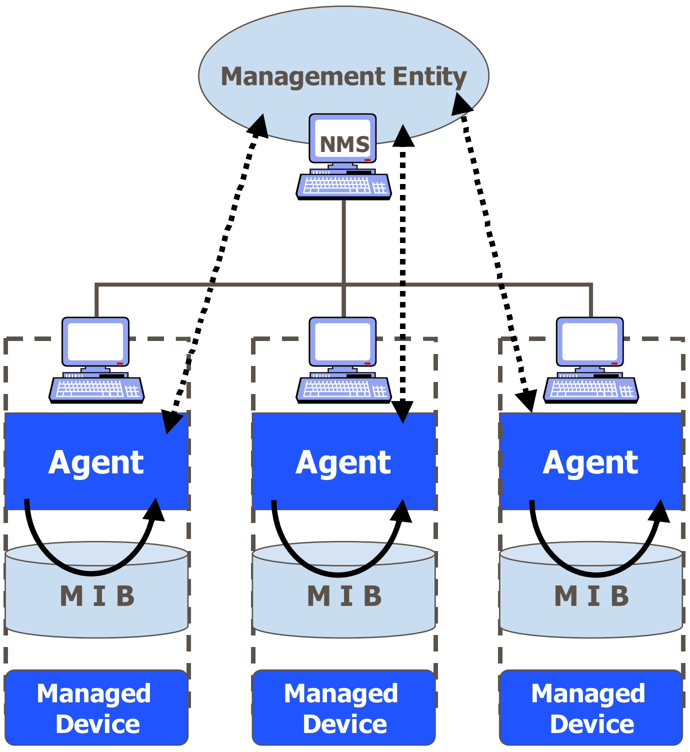


Figure 11. SNMP Managed Network

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 the parameters values upon the request from the NMS, as well as Subscribe mechanism, which deals with asynchronous messages sent by Agent to NMS when a particular event happens.

The Managed Devices monitored and controlled using four basic SNMP commands are **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 also to sequentially gather information in variable tables, such as a routing table.

Management Information Base:

A *Management Information Base (MIB)* is a collection of information that is organised hierarchically. MIBs are accessed using a network-management protocol such as SNMP. They are comprised of managed objects and are identified by object identifiers.

A managed object (sometimes called a MIB object, an object, or a MIB) is one in a number of specific characteristics of a managed device. Managed objects are comprised of one or more object instances, most of which are variables.

Two types of managed objects exist: scalar and tabular. *Scalar objects* define a single object instance. *Tabular objects* define multiple related object instances that are grouped in MIB tables.

The monitoring parameters are strictly hierarchically classified when stored in MIBs. The MIB structure is defined by *SMI* (Structure of Management Information) rules. They define the hierarchy of the managed objects, which are identified by object identifiers. Figure 12 illustrates the object identification. The NMS requests the management objects by their id. For example, the IP object has id 1.3.6.1.2.1.4.

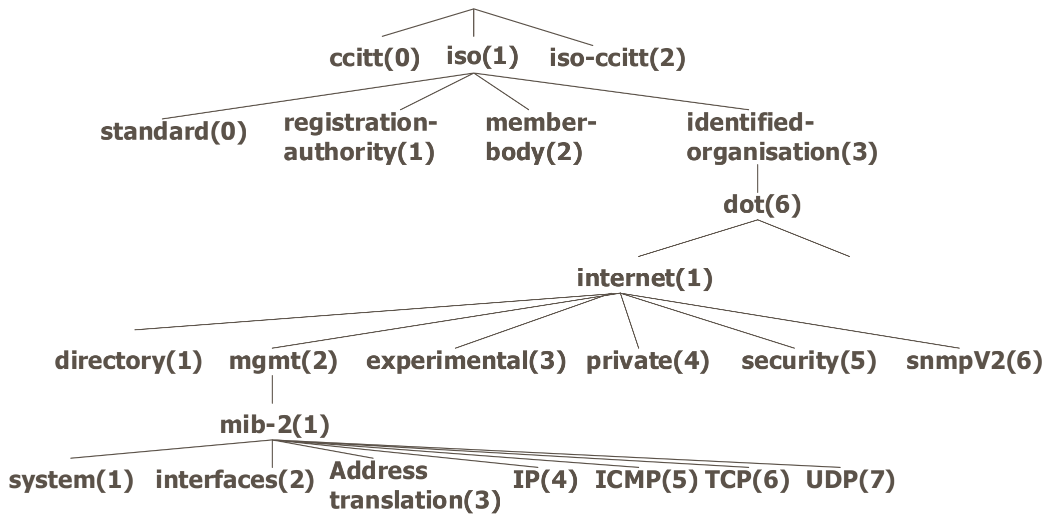


Figure 12. The Structure of the Management Information

SMI (Structure of Management Information) specifies that all managed objects have a certain subset of Abstract Syntax Notation One (ASN.1) data types associated with them. Three ASN.1 data types are required: name, syntax, and encoding. The name serves as the object identifier (object ID). The syntax defines the data type of the object (for example, integer or string). The SMI uses a subset of the ASN.1 syntax definitions. The encoding data describes how information associated with a managed object is formatted as a series of data items for transmission over the network.

SMI also defines several data types as follows:

- **Network Addresses** represent addresses from a particular protocol family SNMPv1. They support only 32-bit IP addresses, but SNMPv2 can support other types of addresses as well.
- **Counters** are non-negative integers that increase until they reach a maximum value and then return to zero. In SNMPv1, a 32-bit counter size is specified. In SNMPv2, 32-bit and 64-bit counters are defined.
- **Gauges** are non-negative integers that can increase or decrease but always retain the maximum value reached.
- **Ticks** represent a hundredth of a second since some event.
- **Opaque** represents an arbitrary encoding that is used to pass arbitrary information strings that do not conform to the strict data typing used by the SMI.
- **Bit strings** are defined only in SNMPv2 and comprise zero or more named bits that specify a value.

Operations:

SNMP is a simple request/response protocol. The network-management system issues a request, and managed devices return responses. This behaviour is implemented by using one of four protocol operations:

- ***Get*** operation is used by the NMS to retrieve the value of one or more object instances from an agent. If the agent responding to the Get operation cannot provide values for all the object instances in a list, it does not provide any values.
- ***GetNext*** operation is used by the NMS to retrieve the value of the next object instance in a table or a list within an agent.
- ***Set*** operation is used by the NMS to set the values of object instances within an agent.
- ***Trap*** operation is used by agents to asynchronously inform the NMS of a significant event. The SNMPv2 Trap operation serves the same function as that used in SNMPv1, but it uses a different message format and is designed to replace the SNMPv1 Trap.

SNMPv2 also defines two new protocol operations:

- ***GetBulk*** operation is used by the NMS to efficiently retrieve large blocks of data, such as multiple rows in a table. GetBulk fills a response message with as much of the requested data as it can fit.
- ***Inform*** operation allows one NMS to send trap information to another NMS and to then receive a response. In SNMPv2, if the agent responding to GetBulk operations cannot provide values for all the variables in a list, it provides partial results.

Security:

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.

<u>Interoperability:</u>

As presently specified, SNMPv2 is incompatible with SNMPv1 in two key areas: message formats and protocol operations. SNMPv2 messages use different header and protocol data unit (PDU) formats than SNMPv1 messages. SNMPv2 also uses two protocol operations that are not specified in SNMPv1. Furthermore, RFC 1908 defines two possible SNMPv1/v2 coexistence strategies: proxy agents and bilingual network-management systems.

A SNMPv2 agent can act as a proxy agent on behalf of SNMPv1 managed devices, as follows:

- An SNMPv2 NMS issues a command intended for an SNMPv1 agent.
- The NMS sends the SNMP message to the SNMPv2 proxy agent.
- The proxy agent forwards Get, GetNext, and Set messages to the SNMPv1 agent unchanged.
- GetBulk messages are converted by the proxy agent to GetNext messages and then forwarded to the SNMPv1 agent.

The proxy agent maps SNMPv1 trap messages to SNMPv2 trap messages and then forwards them to the NMS.

Bilingual SNMPv2 network-management systems support both SNMPv1 and SNMPv2. To support this dual-management environment, a management application in the bilingual NMS must contact an agent. The NMS then examines information stored in a local database to determine whether the agent supports SNMPv1 or SNMPv2. Based on the information in the database, the NMS communicates with the agent using the appropriate version of SNMP.

<u>Products:</u>

The following table aims at an overview of existing management applications both in the public or commercial domains and is not at all intended to give a complete list of existing applications.

Tool	Description	License

Tool	Description	License
MRTG	A Tool that gathers counter values from managed systems and displays these as graphs on WWW pages. This tool is widely used for network load monitoring, logging and statistics gathering. The Tool is using the PERL script language (which has a SNMP built in API) and can therefore be easily integrated on UNIX platforms. The Integration into W2k based systems is also possible.	Open Source
SNMP Browser	A tool that allows browsing a SNMP MIB through a comfortable graphical user interface and displays the gathered values as graphs. This tool was developed at RUS in JAVA and is therefore available on all Java enabled platforms.	Proprietary
NIFTY	An IP traffic flow-monitoring tool. It allows the online monitoring of IP traffic flows i.e. a traffic relation between two BSD sockets. A network manager is able to monitor in a time window of some minutes the duration and packet rate of an existing IP traffic flow and its bandwidth.	Open Source
HP OpenView	A very good tool for network monitoring that enables to control and monitor large scale IP based networks	Commercial (>20.000 €)
SUNNetManager	Same as HP OpenView	Commercial (>20.000 €)

3.2 JAVA MANAGEMENT EXTENSION (JMX)

Java Management Extension [10] 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 for creation of smart Java agents and manageable applications. The JMX components also provide 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)
- SNMPI Manager API (currently reviewed by the Java Community Process)

Architecture: The JMX propose a three-layer architecture comprising:

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

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

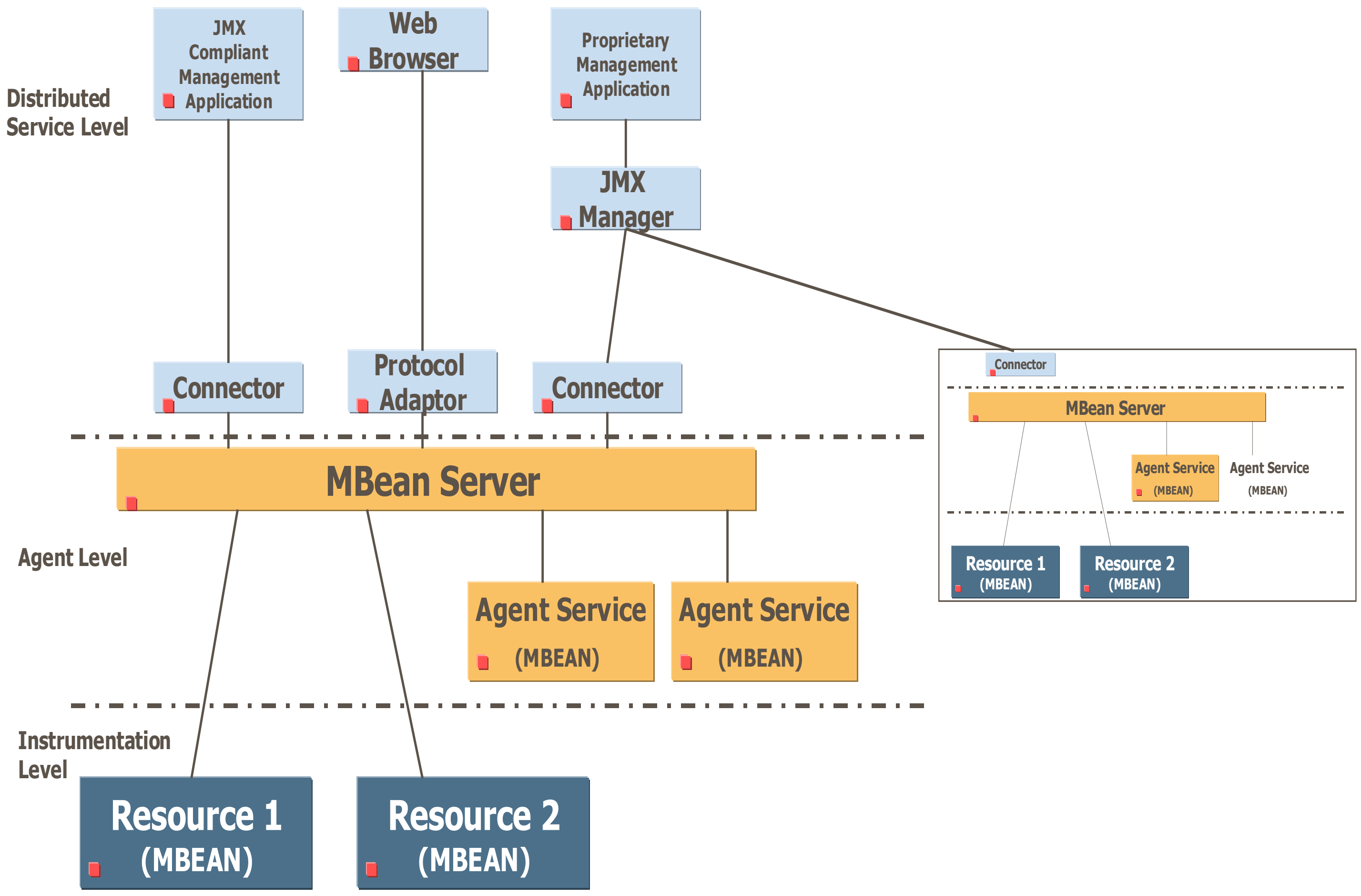


Figure 13. Relationships between components of the JMX architecture

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. Instrumentation can be done through a Java interface or thin Java wrapper, by means 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 MServer. This server manages all communications between the MBeans.

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

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

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

The JMX agents act as an MBean Server handling MBeans using a set of services. Due to this separation, the agent and instrumented resources can be placed on different hosts. Similar to the approach of the instrumentation level, the JMX agent is designed to be independent of Management applications of an upper level.

Distributed Services Level: The blue blocs in Figure 13 represent the Distributed Services level, which deals with Management applications. However, this level is not yet well defined in the JMX specification. This level determines the interfaces for implementation of JMX managers that are purposed to integrate managed resources seamlessly to their environment. In addition, the components named Connector and Protocol Adaptor are used to provide information to different clients.

Companies and Products:

JMX is supported by a number of important and influential companies, including:

- **Computer Associates**
- **GemStone Systems**
- **Inprise Borland**
- **IONA** (leads the process to define the CORBA interface to the JMX specification, since their clients require that both EJB/J2EE and CORBA applications should be administrated from a single management console.)
- **Motorola**
- **Powerware** (provides scalable solutions, as well as enterprise-wide power and foundation management in the distributed computer environment through Java technology.)
- **Schmid Telecommunication**
- **IPlanet**
- **TIBCO Software**
- **AdventNet** (Delivered the first independent implementation of JMX. AdventNet ManageEngine leverages JMX to offer standards-based business-oriented manageability for business processes, applications and middleware infrastructure. AdventNet's JMX technology is widely used by more than 150 customers in various industries.)

The following products passed the compatibility tests and therefore were certified as JMX Compliant:

- ManageEngine, AdventNet
- Agent Toolkit - Java/JMX Edition, AdventNet
- WebLogic Server 7.0, BEA Systems
- SonicXQ and SonicMQ, Sonic Software
- Java Dynamic Management Kit (JDMK), Sun Microsystems

The products below use the JMX technology:

- AdventNet: ManageEngine Applications Manager
- AdventNet: Web NMS
- Alignment Software: AppAssure
- BEA: WebLogic Collaborate
- ComArch: Tytan Mobile Portal

- Computer Associates: Unicenter
- Covalent Technologies: Covalent Application Manager
- Critical Path: System Console
- Hewlett Packard: OpenView
- IBM/Tivoli
- IBM: WebSphere Application Server
- IBM: WebSphere Business Components
- IBM: WebSphere Business Integrator
- IBM: WebSphere Voice Server
- Innovative Systems Design: ITVerify
- The Jakarta Project (Apache): Phoenix
- JBoss
- jNETx: Parlay/OSA Gateway
- Log4j
- Lutris: EAS
- Macromedia: FlashMX
- Sun Microsystems: Java Dynamic Management Kit (Java DMK)
- Sun Microsystems: Netra CT Managed Object Hierarchy (MOH)
- Sun Microsystems: Netra HA Suite
- Sun Microsystems: Netra T1
- Sun Microsystems: DReAM, Distributed Resource Allocation Manager
- Sun Microsystems: Sun Open Network Environment (Sun ONE) Application Server
- Sun Microsystems: Sun ONE Portal Server
- Tomcat

3.3 DISTRIBUTED MANAGEMENT TASK FORCES STANDARDS (DMTF)

Talking about standardisation systems, one cannot avoid DMFT initiatives. Although this organisation works only on management information models, these standards are often used in distributed supervision systems. The most interesting standards are outlined below:

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

- **Desktop Management Interface <http://www.dmtf.org/standards/dmi> (DMI):** These standards generate a standard framework for managing and tracking components in a desktop pc, notebook or server.
- **Directory Enabled Network Initiative <http://www.dmtf.org/standards/den> (DEN):** The Directory Enabled Network (DEN) initiative is designed to provide building blocks for intelligent management by mapping concepts from CIM (such as systems, services and policies) to a directory, and for integrating this information with other WBEM elements in the management infrastructure.
- **Web-Based Enterprise Management <http://www.dmtf.org/standards/wbem> (WBEM):** This initiative is a set of management and Internet standard technologies developed to unify the management of enterprise computing environments.
- **Alert Standard Format <http://www.dmtf.org/standards/asf> (ASF):** This specification defines remote control and alerting interfaces that best serve clients' OS-absent environments.
- **System Management BIOS** <**http://www.dmtf.org/standards/smbios> (SMBIOS):** The SMBIOS Specification addresses the way motherboard and system vendors present management information about their products in a standard format by extending the BIOS interface on Intel architecture systems.

3.4 Existing Commercial Supervision Frameworks

This section presents an outline of the most famous supervision frameworks. The architecture overview contains components description, their purpose, interaction models and supervision approaches.

3.4.1 Unicenter TNG (Computer Associated)

The Unicenter TNG (The Next Generation) is an enterprise management tool, which provides agents for operating system and database management, as well as, user interface facilities.

This tool is highly customisable and extensible thanks to well defined architecture allowing integration of third-party agents and user interfaces.

Architecture:

The architecture of Unicenter TNG provides the means to implement a new paradigm of Business Process Views. Although, it is a fully integrated architecture, three major levels can be identified in it (Figure 14):

- **World View** including Real World Interfaces is a graphical user interface driven by Common Object Repository (CORE). The Core is a central storage mechanism for all components of Unicenter TNG, it is accessible via management functions and third-party applications;
- **Enterprise Management** containing dedicated manager is the Core management facility that provides resource management throughout an enterprise;

- **Agents Factory** comprises all the agents, which are the means to monitor and control all the aspects of a business enterprise.

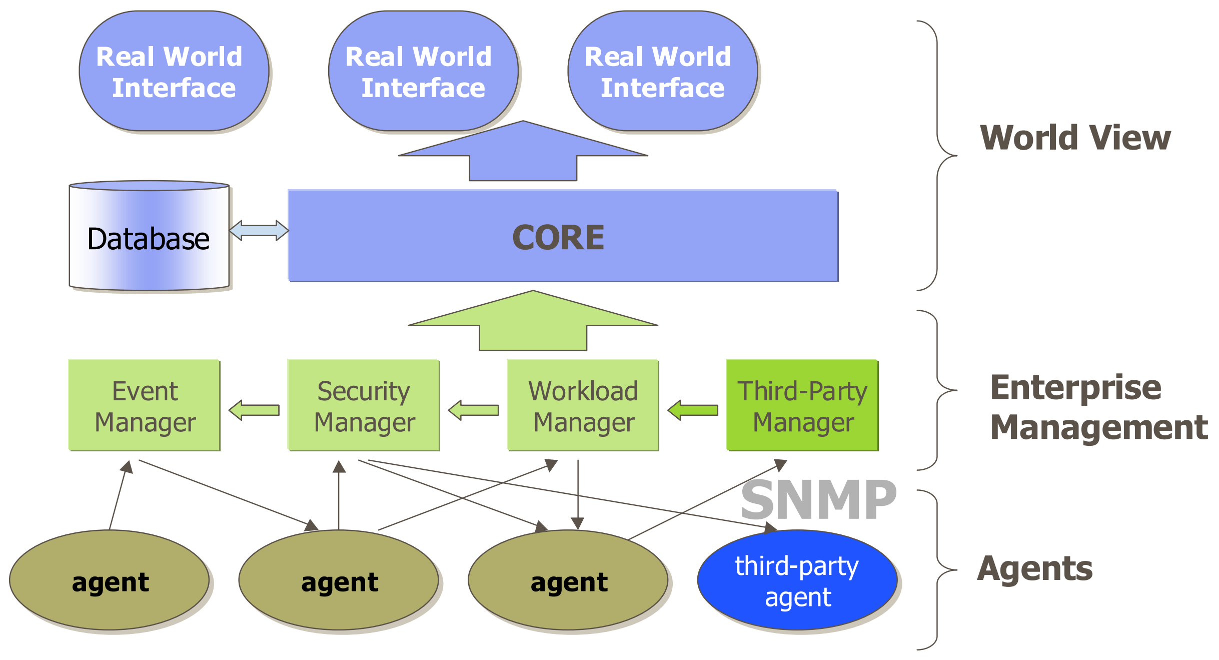


Figure 14. Unicenter TNG Architecture

World View

World View provides a seamless user interface, which is one of the strongest integration capabilities of the Unicenter TNG. It allows each management application to identify the objects it manages as well as the relationships between those objects. This enables the **Real World Interface** to display, navigate, and manage objects from different applications in one seamless environment. When object administration is desired, the management application is invoked in context. Users can also customise the interface, without impacting the behaviour or function of the management applications. The Real World Interface draws on the Common Object Repository to generate management maps dynamically.

The **Common Object Repository** is the location where all management functions store information about the managed objects, their properties, and relationships. Objects for the common management services are also stored in the repository. Search and query capabilities enable management functions to find relevant sets of objects to operate upon. Third-party applications and all Unicenter TNG components access CORE. The CORE is database independent and designed for multi-user and multi-system operations.

Enterprise Management

Enterprise Management is a collection of integrated system management functions available through Unicenter TNG.

Unicenter TNG Enterprise Management permits to manage system resources and to add policy-based automation, security, reliability, and integrity to the environment. This management can be accomplished in a centralised fashion and be distributed throughout the network. It can also use a hybrid approach combining these otherwise different approaches. It allows management at a functional level to best match the unique structure of the business enterprise.

Further, each enterprise management workstation can be customised to reflect the needs of a particular administrator. For example, one workstation can manage backup and archiving of storage, while security workstations can be placed throughout the network, bringing user administration closer to each business unit.

The main components of this level are **Managers**, which 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.

The following are examples that managers can perform:

- Event Management directs the messages to a centralised location and processes them automatically, a task that would otherwise require manual intervention;
- Problem Management records, accesses and tracks problems related to the system and end-user requirements;
- Workload Management automates repetitive or calendar-based processing;
- File Management performs backup and restores operations, and provides integrity and reliability of the information stored on external media;
- Security Management secures the computing environment;
- Performance Management monitors, measures and reports on the usage and performance of the system resources.

Agents

An agent, in the Unicenter terms, is an application that supports network management. An agent typically resides on a managed software node, such as a Windows NT server or workstation, and provides information to a management application (manager).

This information is interpreted according to a management protocol that is understood by both managers and agents.

Both agent and management applications can view the collection of data items for the managed resource. This collection is defined by the management information base (MIB - see section Simple Network Management Protocol (SNMP). Each MIB describes attributes that represent aspects of the managed resource. The network management platform accesses MIB data using SNMP.

Due to open APIs and standard protocols, the 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.

TNG Unicenter’s Distributed Management Approach

In TNG Unicenter’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, then perform analyses and correlations on the data received about their management environment.

For example, Unicenter TNG has the following managers: a workload manager, a storage manager, an asset manager, a problem manager, a software distribution manager, a configuration manager, a file manager, a calendar manager, a report manager, a user/security manager...

There is also a special manager called Distributed State Machine (DSM), which manages 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.

TNG Unicenter 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…

3.4.2 NAGIOS

Nagios [13] is a system and a network monitoring application distributed under the terms of the GNU General Public License. It watches specified hosts and services and is able to alert in case of alarm or state change.

The following features of Nagios include:

- Monitoring of network services (SMTP, POP3, HTTP, NNTP, PING, etc.);
- Monitoring of host resources (processor load, disk usage, etc.);
- Simple plugin design that allows users to develop their own service checks;
- Paralleled service checks;
- Ability to define network host hierarchy using "parent" hosts, allowing detection of and distinction between hosts that are down and those that are unreachable;
- Contact notifications when service or host problems occur and get resolved (via email, pager, or user-defined method);
- Ability to define event handlers to be run during service or host events for proactive problem resolution;
- Automatic log file rotation;
- Support for implementing redundant monitoring hosts;
- Optional web interface for viewing current network status, notification and problem history, log file, etc.

Nagios was originally designed to run under Linux, although it should work under most other Unix systems as well. This also means that, Nagios has been mainly build for monitoring Unix networks.

Nagios is an example of a successful monitoring framework, due to its openness, simplicity, portability and extensibility.

The following section concentrates on its architectural basics.

Architecture:

First of all, Nagios is designed to reside on a single monitoring host. However, it contains a web interface compatible to most of the web-servers that allows remote access. Therefore, Nagios itself consists of the core program logic, a CGI web interface and a set of configuration files. The configuration files contain all monitoring services and hosts and a number of directives that affect the way Nagios operates. Remote hosts and services are monitored by means of specific plug-ins.

Unlike many other monitoring tools, Nagios does not include any internal mechanisms for checking the status of services, hosts, etc. Instead, Nagios relies on external programs (called plug-ins) to implement monitoring and control capabilities. Nagios executes a plug-in when there is a need to check a service or a host that is being monitored. The Nagios concept does not impose any requirements on plug-in monitoring functionality realisation. The only requirements concern the interface to the Core Logic process. The plugin is required to send standard formatted monitoring values to Linux standard output stream. The monitoring parameters description, bounds, status change should also be defined.

Hence, the plug-in performs the check and then returns the results to Nagios. Nagios processes the results received from the plug-in and takes necessary actions (running event handlers, sending out notification, etc).

The Figure 15 below shows separation of plug-ins from the Nagios core program logic.

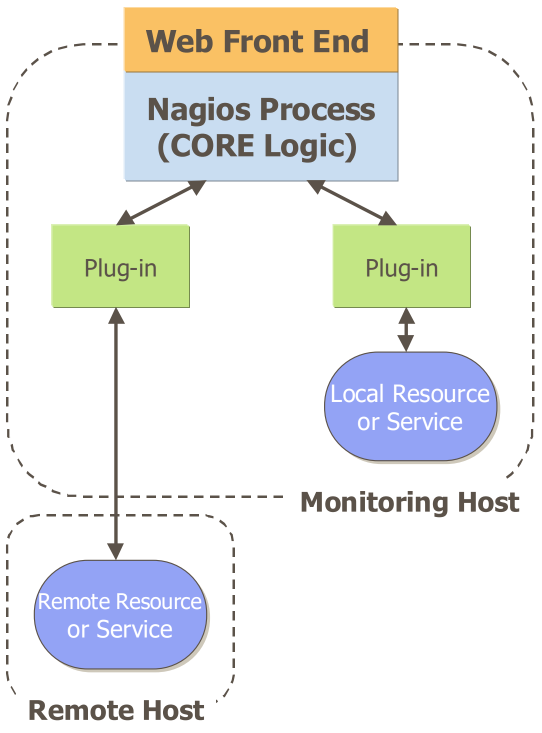

Web Front End
Nagios Process (CORE Logic)
Plug-in
Plug-in
Local Resource or Service
Monitoring Host
Remote Resource or Service
Remote Host

Figure 15. Nagios Plug-in Architecture

Plug-ins are compiled executables or scripts (Perl, shell, etc.) that can be run from a command line.

Nagios executes the plug-ins, which then check local or remote resources or services of some type. When the plug-ins have finished checking the resource or service, they pass the results of the check back to Nagios for processing.

Monitoring Approach:

Passive and Active Service Check

One of the features of Nagios is that it can process service check results that are submitted by external applications. Service checks that are performed and submitted to Nagios by external applications are called **passive checks**. Passive checks can be contrasted with active checks, which are service checks that have been initiated by Nagios.

In contrast to Active checks, Passive checks are useful for monitoring services that are:

- Located behind a firewall, and therefore cannot be checked actively from the host running Nagios;
- Asynchronous in nature and therefore cannot be actively checked in a reliable manner (e.g. SNMP traps, security alerts, etc.).

The only real difference between active and passive checks is that active checks are initiated by Nagios, while passive checks are performed by external applications. Once an external application has performed a service check (either actively or by having received a synchronous event like an SNMP trap or security alert), it submits the results of the service "check" to Nagios through the external command file. The next time Nagios processes the contents of the external command file it will place the results of all passive service checks into a queue for later processing. The same queue that is used for storing results from active checks is also used to store the results of passive checks.

Nagios will periodically execute a service reaper event and scan the service check result queue. Each service check result, regardless of whether the check was active or passive, is processed in the same manner. The service check logic is exactly the same for both types of checks. This provides a seamless method for handling both active and passive service check results.

If an application that resides on the same host as Nagios sends passive service check results, it simply writes the results directly to the external command file as it was outlined above. However, applications on remote hosts cannot perform the same operation so easily. Therefore, there is the NSCA add-on. This add-on allows remote hosts to send passive service check results to the host that runs Nagios. The add-on consists of a daemon that runs on Nagios hosts and a client that is executed from remote hosts. The daemon listens for connections from remote clients, performs some basic validation on the submitted results, and then writes the check results directly into the external command file (as described above).

The **active checks** are well suited for services that lend themselves to periodic checks (availability of an FTP or web server, etc), whereas passive checks are used to handle asynchronous events that occur at variable intervals (security alerts, etc.).

Indirect Service Check

Publicly accessible network services can be monitored directly using plug-ins, for example, Web, POP, SMTP and FTP servers. However, there are "private" resources/services like disk usage, processor load, etc. that are restricted to administrators. To monitor these private resources, the Nagios framework supports Indirect Service Check. This approach involves an intermediary agent, accessible via firewalls using secure protocols. That intermediary agent operates on restricted hosts as an instance of Nagios Core Process accessing locally exposed monitoring information.

Indirect checks are used for:

- Monitoring "local" resources (such as disk usage, processor load, etc.) on remote hosts;
- Monitoring services and hosts behind firewalls;
- Obtaining more realistic results of checks of time-sensitive services between remote hosts (i.e. ping response times between two remote hosts).

3.4.3 HP OPENVIEW

HP OpenView [12] is a supervision framework, which consists of several independent products. Depending on the supervision requirements, these products can be integrated to a powerful solution. These products are:

- IT operations (ITO);
- Node Manager;
- Performance Manager;
- Reporting;
- Etc.

ITO is a software application that provides central operations and problem management for a multivendor, distributed, computing environment. ITO is event and message driven. An ITO managed object typically supports some operations, for example, get value and reset, and is able to emit unsolicited events (status change, error situation). Events which contain some sort of text are called messages; those are the ones ITO is interested in. Messages may originate from different sources and may have different formats. ITO message sources could be:

- Messages sent to the console
- Messages written to application log files
- SNMP traps sent by various applications
- Messages sent through scripts using the ITO open message interface

When a problem occurs or a threshold is reached in the computing environment, an ITO managed object switches to a fault state. Typically, an error message is generated and fed to ITO. Automated or operator-initiated actions can be defined for messages. Alternatively, ITO can be configured to guide the operator to resolve problems by performing corrective actions.

Architecture:

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. Figure 16 illustrates this interaction model.

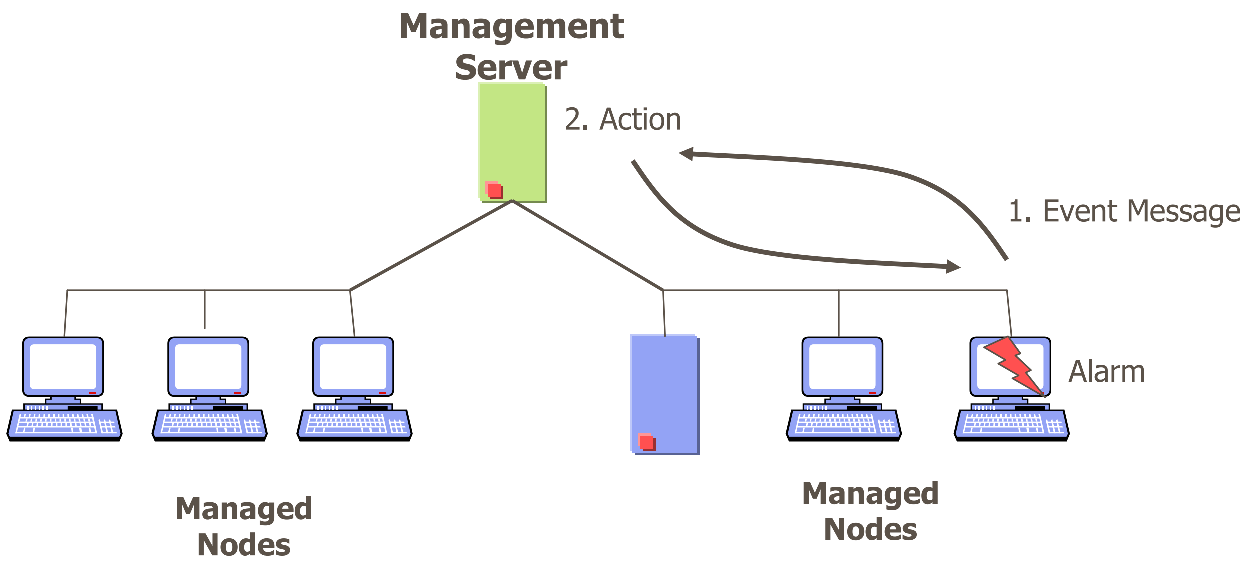


Figure 16. ITO interaction model

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

More precisely, ITO software defines the following two high-level components:

- Management Server;
- Managed Nodes.

Below in the hierarchy, there exist the following 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, manages 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.

The communication between server and nodes is secured by the proprietary DCE (Distributed Computing Environment) implementation. It provides distribution management, as well as security services, standard for DCE.

Integration capability:

Custom agents can be integrated into ITO, at different levels, through various interfaces with the aim 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 log-files, 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, programs based on MIB variables can be started by ITO and use API to return values. The monitored values can then be compared to threshold limits;
- ITO Developer's Toolkit – a C library providing a high-level API to the following communication streams:
    - Server Message Stream;
    - Agent Message Stream;
    - Legacy Link;
    - Application Response;
    - Message Event.

3.4.4 TIVOLI

The **Tivoli Management Environment (TME)** [11] is IBM products line which 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 into 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. According to the TME architecture, all the monitored entities (e.g. hosts) constitute Tivoli Management Region (TMR) that is managed by the main TMR Server (see the next subsection for details). A single system administrator can perform the following tasks for bunches of network systems:

- Manage user and group accounts;
- Deploy new or upgrade existing software;
- Maintain an inventory of the 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.

Architecture:

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

- **TMR Server** – It includes libraries, binaries, data files, and graphical user interface necessary to install and manage a TME environment for the given TMR. 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 communications with the TMR server that the endpoint occasionally requires.
- **Endpoint** – An endpoint is a 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.

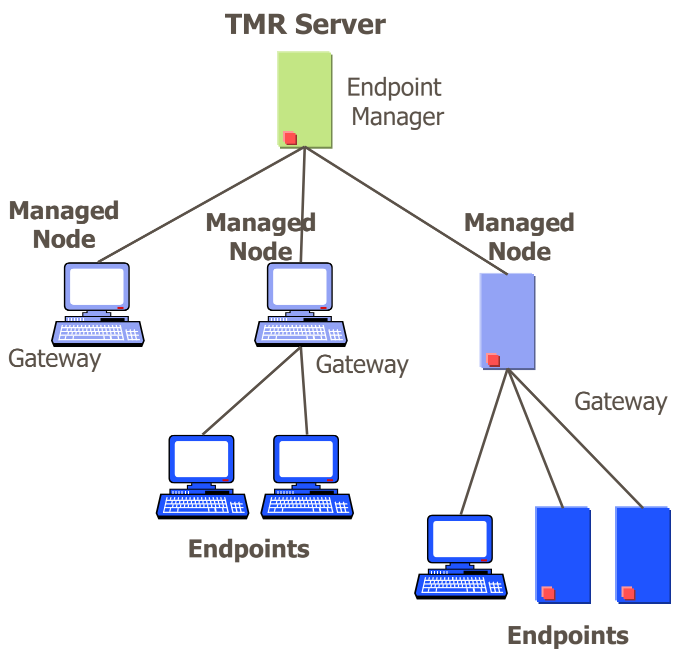


Figure 17. TME Framework Nodes

As depicted in Figure 17, 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 most of the inter-TMR communications take place.

TME provides a distributed environment on top of which a system management application is 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 the distribution of data to multiple targets, Mdist sets up a distribution tree of communication channels from the source host to targets via 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, with all the other communications using the TME Framework built on top of these two communications services:

- **Inter-Object Messaging (IOM):** IOM represents a 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 for 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;
- **Inter-dispatcher communication (objcall service):** 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 is not broken unless the network breaks it, or unless one of the dispatchers is restarted.

3.5 ACADEMIC RESEARCHES

This section outlines current research projects in the domain of distributed supervision systems. They are mainly aimed at application of distributed supervision in particular domains, leaving supervision middleware problems apart. The following list outlines the research projects which contributed to the distributed supervision subject:

ANDROID - The Active Distributed Open Infrastructure Development is an IST project co-funded by the following academic partners: University College London (UK), National Technical University of Athens (Greece) and Universidad Politecnica de Madrid (Spain). The main outcome includes a manageable programmable infrastructure for active networks in the context of IPv6. The following scientific problems were considered:

- Policy Based Resource Management [16];
- Integrity and Security of the Application Level Active Networks [17];
- Weakly Coupled Adaptive Gossip Protocols [18];
- Self-Organizing Resource Discovery Protocols [19];
- Adaptive Security Management [20].

AgentScape - Scalable Resource Management for Multi-Agent Systems. This project provides a platform for large-scale agent systems, supports multiple code bases and operating systems, and interoperability with other agent platforms. It was conducted by the Intelligent Interactive Distributed Systems group of VRIJE Universiteit (Netherlands), which is headed by professor Dr. Frances Braz. The project contributed to the following domains related to distributed supervision:

- Agent Internet-Scale Management [21];
- Grid Management [22].

MANTRIP - The Management Testing and Reconfiguration of IP based networks project is an example of Mobile Agent Technology (MAT) in the context of a Network Management. The academic partners are National Technical University of Athens (Greece), University College London (UK) and University of Surrey (UK). The contributions include:

- Automated Management of IP Networks [23];
- Mobile Agent Security Facility for Safe Configuration [24];
- Quality of Service Management [25].

OPENDREAMS - The “Open Distributed Reliable Environment architecture and Middleware for Supervision” project intended 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 a backbone assuring interoperability and openness of the platform architecture. Ecole Polytechnique Federale de Lausanne (Switzerland) and Polytechnic of Milan (Italy) were the academic partners of the project. They contributed in the following domains:

- Optimisation of object replication [26];
- Architecture for supervision and control systems [27];
- Real-Time CORBA [28].

This brief overview shows a lack of the researches devoted to supervision middleware frameworks. This work is intended to fill in this gap.

3.6 ANALYSIS

The following table outlines a comparison of the parameters defined in the Requirements section for the above-mentioned supervision standards.

Table 1 Distributed supervision standards comparison

Criteria: \ Standard:	***ICMP***	***SNMP***	***JMX***
Interoperability	Full due to protocol level implementation	Full due to protocol level implementation	Full, since implementation is for Java platforms only
Portability	TCP/IP networks	UDP enabled networks	Java enabled platforms only
Flexibility	Fixed number of commands	Simple data types only. Float and compound	Java supports all required types of data including complex

		types are not supported	and compound ones
Intelligence	No intelligence support on protocol level.	No intelligence support on protocol level.	All types of data can be passed including possible intelligence parameters.
Reliability and security	Reliable, no security features	Reliable, the most used SNMP v1 has poor security.	Reliable, security features are provided for by Java services.
Ease of development	Has stable standard implementations for all platforms	Has different APIs for most of the languages: C/C++, Java, etc. No automatic agent generation tools.	Has well defined patterns for agent development.
Ease of deployment	Build-in	Standard client/server installation	Depending on use and realisation, can be build-in(Tomcat server), stand alone or can require application server installation

All this standards are reliable and have useful development facilities. However, network management protocols have some flexibility and security constraints due to their dedication to physical level of management. The only remarkable limitation of JMX is its devotion to Java platform. This constraint could be possibly solved on Distributed Service Level, but, unfortunately this level has not yet been well worked out.

The following table is intended to compare the current supervision frameworks.

Table 2 Supervision frameworks comparison

Criteria: \ Framework:	***Unicenter TNG***	***Nagios***	***Tivoli***	***OpenView***
Interoperability	Secured by SNMP	Great, since the communication between agents is in a text form. However, the agents should run under Linux only.	Only for proprietary Tivoli components.	Secured by SNMP. Proprietary DCE channel has only C interface.
Portability	Multiple SNMP APIs available. Windows and Linux platforms	All kind of programming languages supporting text output, but for Linux platform	Windows and Linux platforms	Multiple SNMP APIs available. Proprietary DCE channel has only C interface.

		only.		Windows and Linux platforms
Flexibility	The supported data types are limited by SNMP	Only text data, of a predefined format, mainly counters	All types of data are supported by proprietary communication mechanism. However custom agent development is difficult.	The supported data types are limited by SNMP Proprietary DCE channel supports all types of data.
Intelligence	Intelligence is supported on agent level. There is no mechanism to pass additional intelligence parameters.	Intelligence is supported on agent level. There is no mechanism to pass additional intelligence parameters.	Additional intelligence parameters can be passed by the Tivoli communication mechanism.	Additional intelligence parameters can be passed by the DCE only.
Reliability and security	Reliable, SNMP dependent security	Reliable, Single host architecture. Security within plug-ins depends on implementation	Reliable, Security is implemented on a full scale.	Reliable, Security is provided only for DCE communication channel, by its services.
Ease of development	Well documented, no automatic agent generation tools	Well documented, no automatic agent generation tools	no automatic agent generation tools, the development is complicated due to proprietary interfaces	Well documented, no automatic agent generation tools
Ease of deployment	Deployment requires sophisticated customisation	Sophisticated configuration files, requires web server installation.	Deployment requires sophisticated customisation	Deployment requires sophisticated customisation

This table shows that the present supervision frameworks have several constraints preventing them to solve the current industrial problems.

The Unicenter TNG major limitation is its usage of the SNMP for all inter-agent communications, which makes it unusable for application supervision requiring compound data transmission. Nagios has a limited portability and restricted message format for communications between plug-ins and the core logic module. The Tivoli has a number of services and agents for commercial standard applications, but its proprietary interfaces makes it difficult to extend the system to a custom application supervision (PDR or DIS-RVM, in this case).

The OpenView framework seems to be most suitable. However, it has a significant disadvantage, i.e. support of APIs for the languages other than C is not provided.

3.7 CONCLUSIONS

To conclude one can point out that there exist various supervision standards that are used in the commercial supervision systems like Tivoli (IBM), OpenView (Hewlett-Packard), Unicenter TNG (Computer Associates). However, most of them have several common constraints, which need to be overcome:

- proprietary interfaces;
- proprietary protocols;
- operating system dependent implementation;
- non-flexible architecture;
- obstinate dedication to a particular commercial application (Oracle, SAP, etc.);
- lack of portability and interoperability.

Although, these supervision systems use open standards (SNMP, JMX, Corba), the above-mentioned constraints complicate integration with third-party monitoring tools to achieve system control at all levels.

Hence, it is necessary to develop a new supervision middleware, which would overcome these constraints on a new level.

Chapter 4. CONCEPT AND IMPLEMENTATION

4.1 PROPOSED SOLUTION

As shown in the state of the art, the protocol based supervision architectures (ICMP, SNMP) have most remarkable interoperability characteristics, i.e. their message format is strictly fixed and they do not impose any limitations on a component implementation, requiring only the protocol support. Consequently, their usage is independent from operating systems and programming languages.

Their force is also their weakness. The strict message format makes it difficult and often impossible to operate with a custom data necessary for modern supervision systems. The network management protocols are inseparable from their transport protocols.

In contrast, the middleware based supervision frameworks like Tivoli or standards like JMX can handle all types of data. However, they also have certain interoperability and flexibility problems.

In the meantime, Web technologies provide with flexible means to build custom, XML based protocols and portable transport mechanisms independent from network protocols (Web Services). This permits to construct solutions that have flexibility in protocol and API.

As a solution this thesis proposes to combine Web technologies to build a supervision middleware that would share advantages of protocol based architectures, i.e. operating system and programming language independency and would provide a flexible and customisable messaging protocol, as well as network portability.

The next section is intended to justify the choice of the Web technologies.

4.2 ANALYSIS OF MIDDLEWARE TECHNOLOGIES

The middleware domain has been developed for more than 20 years [3], [14]. The main goal of the middleware is to hide complexity of distributed systems, in other words, to allow a component to communicate with a distributed system in the same manner as with another single, local component.

There are various middleware concepts that can be logically divided into several groups:

- Remote Procedure Call;
- Distributed Computing Environment;

- Object Oriented Middleware (Java RMI, DCOM, CORBA);
- Component Oriented Middleware (CORBA Component Model, Enterprise Java Beans);
- Agent-Based Middleware (FIPA, ICM);
- Message Oriented Middleware (JMS, Message Queuing);
- Web Services (SOAP, Dot Net Framework, etc.).

This document does not provide a detailed survey of their architecture, which can be found in multiple sources, like [3], [14], [15] and others. Instead, an analysis has been conducted to find the most perspective ones, in regard to the declared requirements for supervision middleware.

Portability and interoperability requirements, are the most difficult ones, and should narrow the solution domain.

Portability is considered as a capability of middleware to run under different operating systems, and over network protocols, in a seamless way. At the same time, interoperability is considered as an ability of components to cooperate even if they run under different operating systems and even if they are coded in different programming languages.

RPC has several implementations, like Cedar RPC and SUN RPC. Its strongest limitation is dedication to procedural model, in this way, interoperability is also limited.

Java-based middleware [10] (Java RMI, Java Beans, Enterprise Java Beans, and Java Message Service) has very portable implementations regarding operating systems, since they all use Java Virtual Machine. However, they have a poor support for other programming languages, even taking into account a possibility to use JNI (Java Native Invocation).

DCOM and .Net Framework [31] are developed to be run only under Microsoft Windows platform. Although there are several open source projects to port .Net on Linux platforms, they are still in an experimental phase. This shows their portability constraints.

FIPA-based agent middleware [32] has several architectural advantages, like well defined autonomous components model, messaging protocol, etc, which can be reused by the supervision systems. Nevertheless, its implementations have poor portability and interoperability characteristics. That makes it difficult to use it in complex heterogeneous systems.

Similarly to FIPA, Message Queuing frameworks are interesting due to their protocol-oriented architecture. Meantime, their messaging protocol differs depending on implementation. Therefore, interoperability is secured only for different components of a single framework.

Contrarily, CORBA [30] and Web Services [42] rely on generic standard transport protocols, GIOP and SOAP. They are quite portable and can operate over different network protocols, like TCP, HTTP, or even FTP and SMTP. Therefore, different implementations of these middleware are interoperable. The both involve component description languages (IDL for CORBA and WSDL for Web Services), that makes it possible to implement components on different programming languages.

CORBA has several advantages in comparison with Web Services. It is an enterprise-focused middleware and provides multiple services:

- **Naming Service** provides an ability to bind a name to an object. It is similar to other forms of directory service;
- **Event Service** supports asynchronous message-based communication among objects. It also supports chaining of event channels, and a variety of producer/consumer roles;
- **Lifecycle Service** defines conventions for creating, deleting, copying and moving objects;
- **Persistence Service** provides a means for retaining and managing the persistent state of objects;
- **Transaction Service** supports multiple transaction models, including mandatory "flat" and optional "nested" transactions;
- **Concurrency Service** supports concurrent, coordinated access to objects from multiple clients;
- **Relationship Service** supports the specification, creation and maintenance of relationships among objects;
- **Externalisation Service** defines protocols and conventions for externalising and internalising objects across processes and across ORBs.

In addition, Corba is more mature and has multiple implementations. It proved its reliability for mission critical applications.

Web Services is a fast developing domain. It gained its popularity due to the usage of XML-based protocols. These protocols are self-descriptive, which means that they can be easily processed by both humans and computers. In addition, XML grammar is very flexible and supports representation of complex data types and structures (enumerations, arrays, lists, hash maps, choices, and sequences) that are required by the supervision systems. Therefore, there are various toolkits implementing Web Services for more than 20 programming languages, including scrip languages (PHP, Perl, etc) required for modern web application management.

As for facilities, Web Services involves Directory Servers purposed to store service location and description. Synchronous and asynchronous communication models are supported. Multiple security mechanisms are available for transport and messaging protocols. Besides, various tools for automatic source code generation are provided for different programming languages.

This all makes CORBA and Web Services to be the most interesting candidates for the supervision middleware to be based on.

Comparing these two technologies, CORBA, as many other traditional distributed computing technologies, tightly couples the remote resource location to the client stub and often requires end-to-end control of the network. On the contrary, Web Services have more robust and transparent architecture, allowing resources (components) to discover each other over wide area networks, which is often necessary for complex distributed applications (see Introduction for examples). Additionally, Web-Service architecture is designed for autonomous resources (components), which is one of the mandatory characteristics of software agents used in supervision systems.

The following tables summarise the outlined properties of the middleware technologies in regard to above-mentioned requirements.

Technology: \ Criteria:	***Portability***	***Interoperability***
RPC	Limitated by procedural model.	No interoperability between implementations.
Java Middleware (RMI, EJB, JMS)	Portable for all operating system supporting Java Virtual Machine.	Java components only.
Microsoft Middleware (DCOM, .NET)	Limited to Windows platform.	High Interoperability between components written on different programming languages (Microsoft only).
FIPA Agent middleware	Several implementations for Linux, Windows and Java-enabled platforms.	Lack of interoperability between implementations.
MOM: Message Queuing Systems	Several implementations for different platforms.	Lack of interoperability between implementations.
CORBA	Several implementations for different platforms and network protocols.	Implementations are interoperable due to GIOP.
Web Services	Several implementations for different platforms and network protocols.	Messaging Protocol oriented architecture, with standard message format. The Implementations are interoperable. [51]

Criteria: \ Technology:	***CORBA***	***Web Services***
General Communication Protocol	GIOP	SOAP
Component Description Language	IDL	WSDL
Communication Mechanisms	Synchronous/Asynchronous Asynchronous communication is secured by Notification Service	Synchronous/Asynchronous
Directory Service	Name Service	Service Registry often secured by UDDI service
Resource distribution	Client subs should be tight to remote resource	Designed for distribution over wide area networks
Components Autonomy	low	high

Maturity	+++++	++

Taking into account all the properties of both technologies (CORBA and Web Services), the present Ph. D. work intends to investigate applicability of Web Technologies and, in particular, Web Services to Supervision Middleware domain.

The chapters below describe the proposed architecture in details. This description starts with a basic overview of applicable Web technologies with the intention to introduce conceptual aspects of the suggested architecture.

4.3 APPLICABLE WEB TECHNOLOGIES

4.3.1 XML

XML (eXtensible Markup Language) is descriptively identified in the XML 1.0 W3C Recommendation [33] as a simple dialect (or 'subset') of SGML. Its goal "is to enable generic SGML to be served, received and processed on the Web in the way that is now possible with HTML". For this reason, "XML has been designed for ease of implementation and for interoperability with both SGML and HTML."

XML is known to have helped to solve some difficult problems in data interchange, such as data integrity. Data integrity can be ensured through schemas and validations, application-specific business rules and internationalisation by requiring unicode in a variety of XML-based languages, e.g. following ones:

- Web Services Conversation Language (WSCL);
- Web Services Description Language (WSDL);
- Web Services for Interactive Applications (WSIA);
- Web Services User Interface (WSUI) Initiative;
- Web Services for Remote Portals (WSRP);
- Web Services Experience Language (WSXL);
- Business Process Execution Language (BPEL) as successor of WSFL & XLANG.

XML is successful because it is a way to describe hierarchies. Anything that can be represented in a tree can be represented in XML. XML is a simple, unambiguous meta-language for describing arbitrarily complex, hierarchical relationships. The hierarchical data description is well understood both in culture and development circles and XML can express parent, child and sibling relationships, requiring little explanation, using any tag and any language.

One of the main advantages of XML is that programmers can separate syntax (data presentation in XML documents) and semantics (data structure), data storage (content of an XML document) and data processing. The term "processing" means various ways of data use in XML applications (data extraction, data change, data presentation in "human-readable" form, etc.).

4.3.1.1 XML SYNTAX

XML document is the text file containing the set of markup tags.

Each element can have content (between the start <…> and end tag </…>) and attributes (within the start-tag). While a single element can look like <…/> (<example/>).

To define tag sets, users must create a Document Type Definition (DTD) that formally identifies (for the appropriate XML document): the elements, their attributes, the relationships between the various elements. The DTD syntax is the part of the general XML standard (see [33]).

XML is more than a language, it is a meta-language [34] in the following sense: different user-defined DTDs establish different semantics over the set of well-formed XML documents.

4.3.1.2 XML SCHEMA VS. DTD

The XML DTD was specified in the XML 1.0 recommendation, published before Namespaces in XML 1.0. The XML DTD ignores the notion of namespace and lacks the flexibility necessary to support them in a simple way. The XML DTD is also a descendant of the SGML DTD, which had been designed for document-oriented applications and lacks a complete type system - a requirement for data oriented applications.

The W3C had a choice between updating the specification of the DTD and creating a new specification; the committee chose to start anew.

In contrast to DTDs, schema documents are built in XML itself [35]. Validation using schemas requires two documents: the schema document and the instance document.

The schema document is the document containing the structure and the instance document is the document containing the actual XML data. The simplest way for an application to determine the schema for an instance document is to use attributes and namespaces to point to an external schema document (*.XSD file), instead of using DOCTYPE declaration to point to an external DTD file.

XML Schema offers a powerful set of tools for defining acceptable document structures and content. XML Schema is an alternative to DTDs as far as describing and validating of data in an XML environment are concerned. In addition, it enables developers to create precise descriptions with a richer set of data types – such as booleans, numbers, currencies, dates and times essential for today's applications.

Schemas are rather more powerful than DTD, but that power comes with substantial complexity.

Nevertheless, the XML Schema approach provides much more capabilities than DTD does. There are thirteen (13) kinds of components in all. The primary components, which may (type definitions) or must (element and attribute declarations) have names are as follows: simple type definitions, complex type definitions, attribute declarations, element declarations.

Forty-two (42) simple types are defined as a part of the recommendation, including string, int, date, decimal, boolean, timeDuration, and uriReference. One can also create new types and data structures.

4.3.1.3 XML PROCESSOR (PARSER)

The next basic concept of XML technology is so called XML processor (or parser). The point is that XML document itself is no more than a text file. The XML technology introduces a concept of standard component (application) intended:

- to read and to examine (to parse) XML document in accordance with the rules of XML syntax;
- to check for compliance with the structure defined in an appropriate DTD (if this DTD is declared within the XML document).

If the XML document (and DTD) is compliant with syntax standard, namely:

1. an XML processing instruction identifying the version of XML being used, the way in which it is encoded, and whether it references other files or not, e.g. <?xml version="1.0" encoding="UTF 8" ?>;
2. the document is "fully-tagged" and consists of a root element, within which all other mark-ups are nested,

then such a document is called well formed.

If, in addition, an XML document structure satisfies the given DTD or schema, then such a document is called valid.

The aim of the XML processor is to check if the document processed is well formed and valid.

There are two main kinds of XML processor: SAX (Simple API for XML) parser [36]; DOM (Document Object Model) parser [37].

In an event-based API like SAX, the parser sends events to a listener to process or ignore them. While in a tree-based API like DOM, the parser builds a data tree in memory.

SAX analyses an XML stream as it goes by, much like a stream input.

The SAX API allows a developer to capture these events and act on them.

SAX processing involves the following steps:

- Create an event handler;
- Create a SAX parser;
- Assign an event handler to a parser;
- Parse a document by sending each event to a handler.

The advantages of this kind of processing are much like the advantages of streaming media. In other words, analysis can get started immediately, rather than having to wait for all of the data to be processed. Besides, because the application is simply examining the data as it goes by, it does not need to store it in the memory. This is a huge advantage when it comes to large documents. In general, SAX is also much faster than the other alternative, the DOM parser.

On the other hand, because the application is not storing the data in any way, it is impossible to make changes to it using SAX, or to move "backward" in the data stream.

The Document Object Model is an API to process valid well-formed XML documents. It defines a logical structure of documents and the way a document is accessed and manipulated. In the DOM specification, the term "document" is used increasingly in a broad sense. XML is used as a way of representing many different kinds of information that may be stored in diverse systems and much of this would traditionally be seen as data rather than as documents. Nevertheless, XML presents this data as documents and the DOM may be used to manage this data.

As for the W3C specification, one important objective for the Document Object Model is to provide a standard programming interface that can be used in a wide variety of environments and applications. The DOM is designed to be used for any programming language. In order to provide a precise, language-independent specification of the DOM interfaces, it has been chosen to define the specifications in OMG IDL, as defined in the CORBA 2.3.1 specification.

With the Document Object Model, programmers can build documents, navigate their structure and add, modify or delete elements and content. Any information stored in an XML document can be accessed, changed, deleted using this model.

DOM is a "traditional" way of handling XML data, by means of which data is loaded into memory in a tree-like structure.

DOM, and by extension tree-based processing, has several advantages. First, as the tree is persistent in memory, it can be modified. Hence, an application can make changes to the data and the structure. It can also work its way up and down the tree any time, as opposed to the stream handling of SAX. DOM can also be much simpler to use, than SAX.

On the other hand, there is a lot of overhead involved in building these trees in memory. It is not unusual for large files to completely overrun a system's capacity. In addition, creating a DOM tree for big XML documents can be a very slow process.

There exists a wide choice of different XML processor implementations in Java, C, Perl and other programming languages. The list of the available free implementations may be found on http://www.garshol.priv.no/download/xmltools/cat_ix.html#SC_XML (more then 30 items). In addition, the following table presents the best-known APIs.

API: \ Criteria:	**Prog. Language**	***XML version***	***XML schema support***	***Parsers***	***Operating Systems***
Xerces2 Java	Java	1.0	1.0	SAX, DOM	Java platforms
Xerces-C++	C++	1.0	1.0	SAX, DOM	Unix, Linux, Win32, Mac OS, OS/390
Xalan-C++	C++	1.0	1.0	SAX, DOM	Unix, Linux, Win32
Xalan-J++	Java	1.0	1.0	SAX, DOM	Java platforms
Microsoft XML parser	Any supporting COM interface	1.0	1.0	SAX, DOM	Win32

SUN JAXP	Java	1.0	1.0	SAX, DOM	Java platforms

4.3.2 WEB SERVICES

The definition of a Web Service according to [42] is

"A Web service is an interface that describes a collection of operations that are network accessible through standardised XML messaging. A Web service is described using a standard, formal XML notion, called a service description. It covers all the details necessary to interact with the service, including message formats (that detail the operations), transport protocols and locations. The interface hides the implementation details of the service, allowing it to be used independently of the hardware or software platform on which it is implemented and also independently of the programming language in which it is written."

The main properties of a Web Service are as follows:

- Language independent;
- Self describing;
- Communication by means of XML.

If applications are built on Web Services they consist very likely of multiple distributed components that are loosely coupled and implemented in different languages. Web Service components are not tightly connected to a specific application. However, they can be used in multiple applications and can themselves employ other Web Service components for fulfilling their task.

The web service architecture defines the three major roles:

- Service provider;
- Service registry;
- Service requestor.

A service provider hosts an implementation of the service. It defines a service description that is published to a Service Registry. Service Requestors can query the service registry in order to find a matching service. If such a service is located, the Service Requestor directly calls the methods offered by the Web Service hosted by the Service Provider. The operations and the involved protocols are shown in Figure 18.

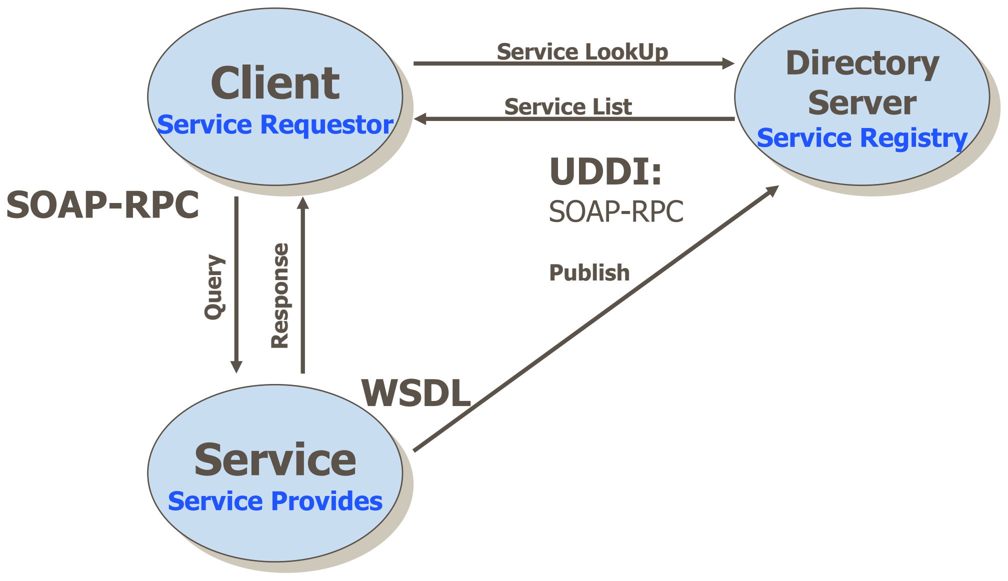


Figure 18. Roles within Web Service Architecture

The **Service**, also called **Service Provider**, is the platform that hosts the implemented Web Service and offers it to the Service Requestors.

The **Client**, also referred to as **Service Requestor**, needs a specific functionality from a third party. It uses the Service Registry to locate this service, and then employs the found Service. The Service Requestor can be a client application of any kind or another service of a higher complexity that uses another Web Service in order to fulfil its task.

The **Directory Server** plays role of a **Service Registry.** It is a searchable storage where service providers publish their service descriptions. Service requestors find services and obtain binding information in the service descriptions. This information is used during development for static binding or during execution for dynamic binding.

For statically bound service requestors, the service registry is an optional role in the architecture, because a service provider can send the description directly to service requestors. Similarly, service requestors can obtain a service description from other sources besides a service registry, such as a local file.

4.3.2.1 Architecture Overview

The Web Service Protocol Stack is organised in multiple layers as depicted in Figure 19.

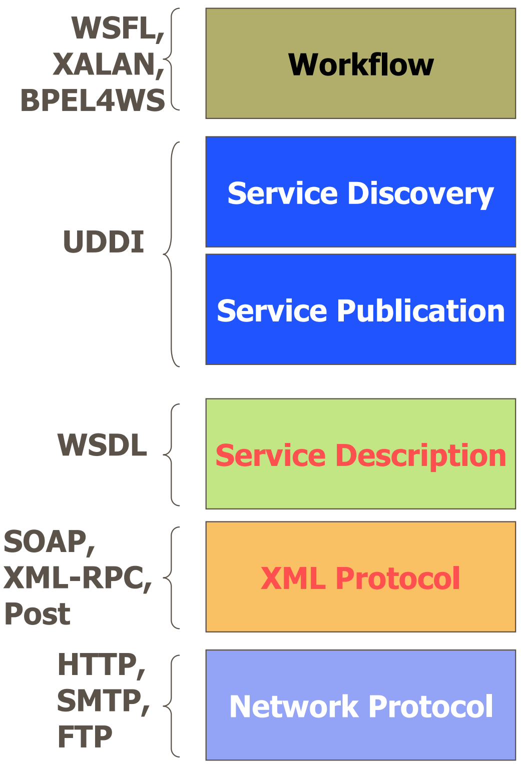


Figure 19. Web Services Protocol Stack

The lowest layer, the network layer, offers basic transport functionality for messages sent between entities. The protocol layer can be any network protocol, no matter stateless or stateful, synchronous or asynchronous. This allows a wide range of well-established Internet protocols to be used including the Hypertext Transfer Protocol (HTTP) and the Simple Mail Transport Protocol (SMTP). On top of these protocols the Simple Object Access Protocol (SOAP) builds the basis for a communication between distributed components. This protocol enables other components and clients to execute methods on remote objects similar to the Internet Inter ORB Protocol (IIOP) and the Simple Remote Method Protocol (SRMP) used by CORBA and RMI. Another major aspect of the Web Service architecture is the service description that enables dynamic binding to self-describing services. The service description can be performed in the Web Service Description Language (WSDL). In order to obtain this description another protocol is necessary for Service Discovery and Publication. The solution used by Microsoft, IBM and others is the Universal Description, Discovery and Integration Protocol (UDDI). However, in principle, other mechanisms such as publication of the WSDL on a web server are also possible. With the above-presented protocol, a complete chain of distributed components can communicate and realise complex workflows and applications.

In common practise, it often happens that complex services mutually cooperate with one another. This cooperation is implemented using languages that describe orchestration between simple and more complex services. The Web Services Flow Language (WSFL) or XLANG serve as examples of such languages.

4.3.2.2 Network Layer

The base layer of the Web Services stack is the network. This layer can be implemented using any number of existing network protocols such as SMTP, FTP, HTTP, RMI, IIOP or others. One of the major advantages of the Web Service architecture is that the service developers do not need to care about this underlying network protocol, as it is completely transparent to the upper layers.

Despite a wide choice of network protocols that can be used, most Web Service applications rely on HTTP. HTTP is widely deployed and is very likely the ideal choice for Internet based applications. For applications with a limited audience for example within an Intranet or specific requirements including security, performance and reliability other protocols could be a better choice.

Hypertext Transfer Protocol (HTTP):

HTTP [46] is a simple stateless protocol that sends its request and response protocol data units over TCP/IP. An HTTP client establishes a TCP connection with an HTTP Server (usually port 80 is used). On this established connection, the HTTP client sends its HTTP Request. The server sends back a response to the client on the same channel. Both the request and response messages can contain arbitrary payload information, typically tagged with the Content-Length and Content-Type HTTP headers. An example of an HTTP Request is shown in Figure 20.

```
POST /foo HTTP/1.1
Host: 192.168.1.1
Content-Type: text/plain
Content-Length: 25

Example of a POST Request
```

Figure 20. Example of an HTTP Request

The HTTP header consists of a plain text. The payload is of the type specified in the Content-Type section of the header. In the example the payload is also in the format of a plain text. A potential answer must start with a return code indicating the type of result and optionally additional information.

```
200 OK
Content-Type: text/plain
Content-Length: 21

Example of a Response
```

Figure 21. Example of an HTTP Response

As shown in Figure 21, a potential Response of the server in case of success could be organised. A response contains also a header and optionally a content body. A Response starts with a status code indicating the type of response and a descriptive text for this kind of answer. Other examples for HTTP responses are shown in Figure 22 and Figure 23.

```
400 Bad Request
Content-Length: 0
```

Figure 22. Example of an HTTP Error Response

```
307 Temporary Moved
Location:
192.168.1.3/bar
Content-Length: 0
```

Figure 23. Example of an HTTP Redirect Response

4.3.2.3 XML PROTOCOL LAYER – SOAP

This layer is responsible for transport of method calls, its parameters and also the results of operations. This layer is in most implementations of a Web Service stack realised by means of the Simple Object Access Protocol (SOAP) (refer to [47]). The SOAP Protocol is expressed using XML and defines within its specification how SOAP messages are transported using HTTP and SMTP. Other mappings are subject to further development in the future.

SOAP offers different kinds of communication types between a service requestor and a service provider. There exists a variety of client/server relations in a synchronous or asynchronous ways. Moreover, one-way messaging (with no response) and notification (push-style response) are also possible (Figure 24).

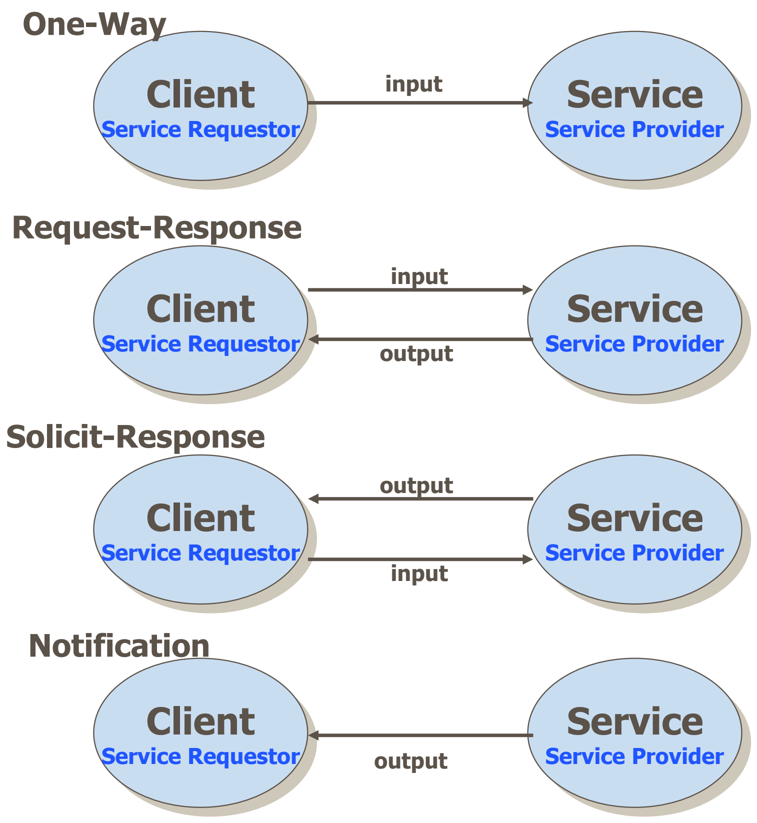


Figure 24. SOAP Communication Types

A SOAP message consists of several logical components. The Object Endpoint ID uniquely identifies the destination of the request. The interface and method identifier describes a method that should be called. The Extension Headers are headers of the SOAP message and the Parameter DATA is the Body of the SOAP message. Figure 25 shows mapping from logical components to a SOAP message transported via HTTP.

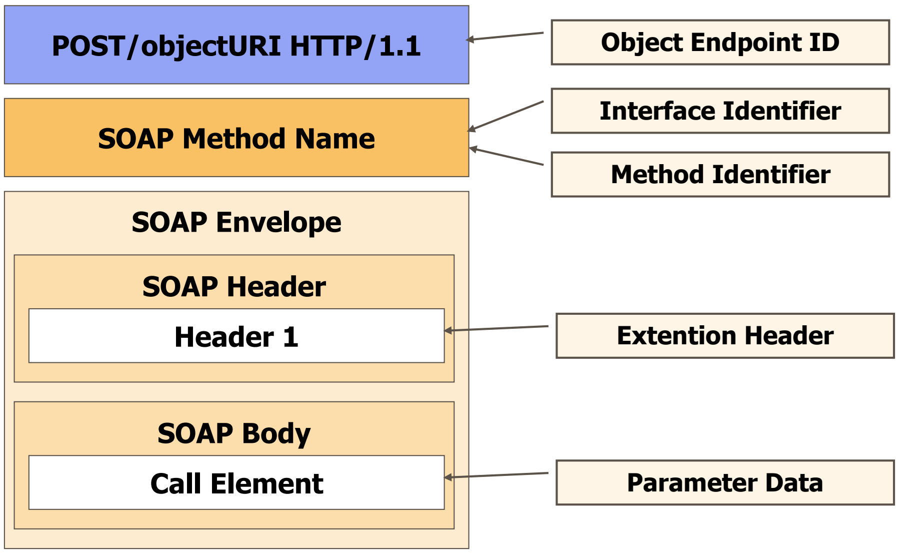
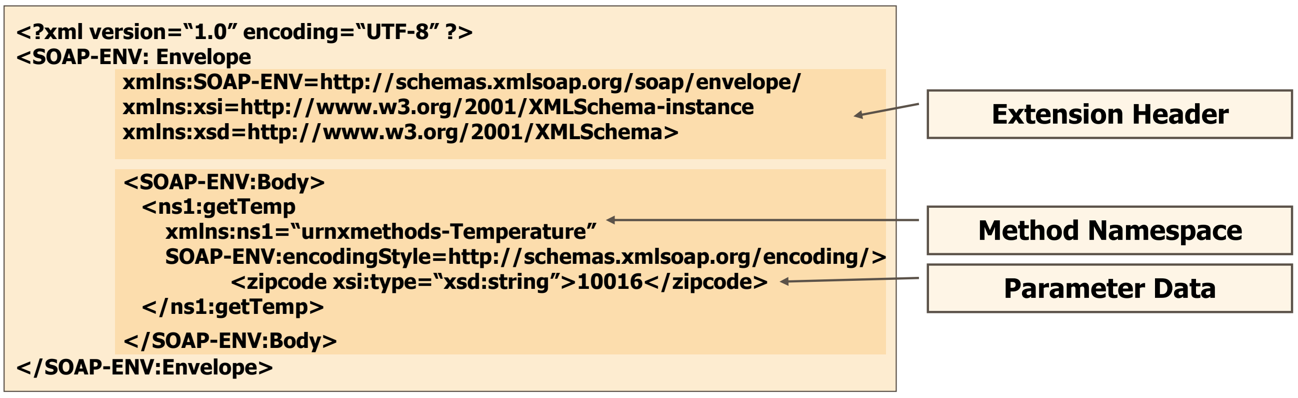


Figure 25. Logical Components of SOAP Message

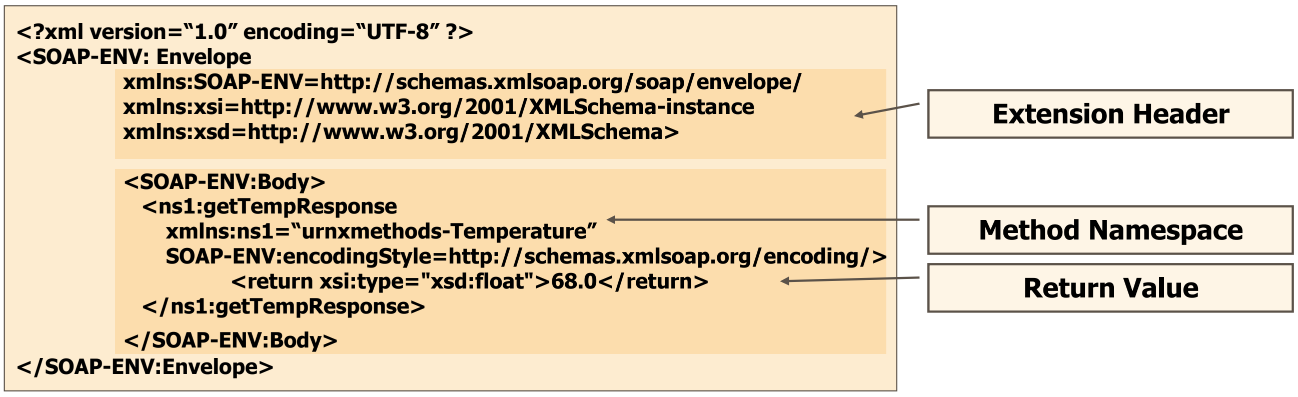
As an example of a SOAP message, Figure 26 shows a message sent to a weather service provided by Xmethods (http://www.xmethods.org), which hosts several Web Services for public use. The weather service provide a temperature parameter for a location indentified by a zip code (Weather-Temperature service). The following figures illustrate SOAP request and response messages.



```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV: Envelope
            xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
            xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
            xmlns:xsd=http://www.w3.org/2001/XMLSchema>

            <SOAP-ENV:Body>
              <ns1:getTemp
                xmlns:ns1="urnxmethods-Temperature"
                SOAP-ENV:encodingStyle=http://schemas.xmlsoap.org/encoding/>
                       <zipcode xsi:type="xsd:string">10016</zipcode>
              </ns1:getTemp>
            </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 26. Example of a SOAP Request message



```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV: Envelope
            xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
            xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
            xmlns:xsd=http://www.w3.org/2001/XMLSchema>

            <SOAP-ENV:Body>
              <ns1:getTempResponse
                xmlns:ns1="urnxmethods-Temperature"
                SOAP-ENV:encodingStyle=http://schemas.xmlsoap.org/encoding/>
                       <return xsi:type="xsd:float">68.0</return>
              </ns1:getTempResponse>
            </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 27. Example of a SOAP Response message

4.3.2.4 Web Service Description Language (WSDL)

The Web Services Description Language (WSDL) [48] serves to explicitly define all interfaces between a service provider (Web Service itself) and service requestors (Clients). A service description in WSDL can be considered as a “communication” contract between these two parties. It concerns the following important parameters:

- interface of a service, which is expressed through its public methods;
- data types for all message requests and responses;
- information about transport protocols;
- information for locating a specified service.

The WSDL definition is similar to a definition in CORBA Interface Description Language (IDL). WSDL is also platform- and language independent and is used primarily (but not limited to) to describe SOAP based services. Besides, WSDL allows a client to locate a service and to invoke the public methods provided by its interface. Moreover, some of WSDL aware toolkits allow execution of remote methods without writing any additional program code.

The WSDL specification defines an XML grammar, which is divided into the following major sections:

- **definitions -** The definitions element is the root element of a WSDL document. It defines global elements for the service itself (e.g. the name of the service) and declares the namespaces that will be used for the rest of the document;
- **types -** As part of the methods parameters or return values of methods, not only simple data types can be used but also complex data types. This section contains a description of these complex data types that may be referred to in other sections. The WSDL is not limited by any specific typing system, but it is a common practice to use the W3C XML Schema for this purpose;
- **message -** Within this section all request and response messages are described. For each message, a separate element is required. A message contains at least the name of the message and optionally message part elements. The message part elements can refer to message parameters or message return values;
- **portType** -These elements group multiple message elements to one-way or round-trip operations. Typically a portType defines more than one operation;
- **binding** - This section contains all the information necessary to specify the way messages are transported over the network. Since WSDL was designed with SOAP in mind, a special extension element service offers the possibility to specify SOAP specific information;
- **service –** This block describes deployment properties of the Web Service.

In order to get a more detailed view on the elements of the WSDL specification, the following section provides a WSDL definition for the Weather-Temperature example, mentioned in the previous section (http://www.xmethods.org).

Figure 28 starts the description with a structure of WSDL definition in XML.

```
<?xml version="1.0" ?>
<definitions>
            <types> This section describes custom types if any</types>
            <message> Each message is describe between these tags </message>
            <portType> This section defines service operation,
                        including pairs of input/output messages </portType>
            <binding> This section declares communication paradigms,
                        including Extension Headers for each message</binding>
            <service>This part describes service deployment parameters</service>
</definitions>
```

Figure 28. WSDL definition structure

Figure 29 depicts the **definitions** block for the Weather-Temperature example.

```
<definitions
            name="TemperatureService"
            targetNamespace="http://www.xmethods.net/sd/TemperatureService.wsdl"
            xmlns:tns="http://www.xmethods.net/sd/TemperatureService.wsdl"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
            xmlns="http://schemas.xmlsoap.org/wsdl/">
...
</definitions>
```

Figure 29. WSDL:Example of a <definitions> block

The parameters of the <**definitions**> tag declare the name of the current service and several namespaces that will be used throughout the rest of the WSDL document. **targetNamespace** parameter defines the current namespace, thus all data (messages, portTypes, bindings) defined in the document will belong to the "http://www.xmethods.net/sd/TemperatureService.wsdl" namespace. **xmlns:tns** parameter indicates a shortcut for the target namespace, hence all variables and types can be called using the **tns** prefix. **xmlns:xsd**, **xmlns:soap** and **xmlns** are declarations of the main namespaces indicating usage of XMLSchema, SOAP and WSDL.

The below example (Figure 30) defines different types of messages, including their content. Here, the **zipcode** represent an input parameter whereas **xsd:string** type indicates usage of a standard XML Schema type, which is an equivalent of the **char*** type in the ANSI C notation. **return** has **xsd:float** type, which is a representation of the **float** type in the ANSI C notation.

```
<message name="getTempRequest">
            <part name="zipcode" type="xsd:string" />
</message>
<message name="getTempResponse">
            <part name="return" type="xsd:float" />
</message>
```

Figure 30. WSDL:Example of a <message> block

In Figure 31, the **portType** block defines the main service operation **getTemp**, which is defined by its input and output messages. Here, the **tns:** prefix in the message parameters indicates that **getTempRequest** and **getTempResponse** are declared in the current document. In addition, in this block, the messages are actually associated with input/output parameters.

```
<portType name="TemperaturePortType">
            <operation name="getTemp">
                        <input message="tns:getTempRequest" />
                        <output message="tns:getTempResponse" />
            </operation>
</portType>
```

Figure 31. WSDL:Example of a <portType> block

The missing information concerns the means to invoke these methods using SOAP. This specification is provided within the binding section (Figure 32).

```
<binding name="TemperatureBinding" type="tns:TemperaturePortType">
            <soap:binding style="rpc"
                        transport="http://schemas.xmlsoap.org/soap/http" />
            <operation name="getTemp">
                        <soap:operation soapAction="" />
                        <input>
                                    <soap:body use="encoded"
                                                namespace="urn:xmethods-Temperature"
                                                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
                        </input>
                        <output>
                                    <soap:body use="encoded"
                                                namespace="urn:xmethods-Temperature"
                                                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
                        </output>
            </operation>
</binding>
```

Figure 32. WSDL:Example of a <binding> block

The binding section describes transport means (in this case Remote Procedure Call - RPC) and for each operation, it defines additional information to be placed in the request/response messages (compare with Figure 26 and Figure 27).

Finally, Figure 33 shows the parameters intended to facilitate the service location.

```
<service name="TemperatureService">
    <documentation> Returns current temperature
                        in a given U.S. zipcode</documentation>
    <port name="TemperaturePort" binding="tns:TemperatureBinding">
        <soap:address location="http://services.xmethods.net:80/soap/servlet/rpcrouter" />
    </port>
</service>
```

Figure 33. WSDL:Example of a <service> block

The service section lists deployment information and gives a readable description for the service within <**documentation**> tags.

In order to simplify understanding of WSDL definitions, 3rd party companies provide specialised tools. The most used one is XML Spy by ALTOVA (http://www.altova.com/). This tool provides a diagram that depicts all major items of the WSDL document and their interconnections in a useful form, as it is shown below in Figure 34.

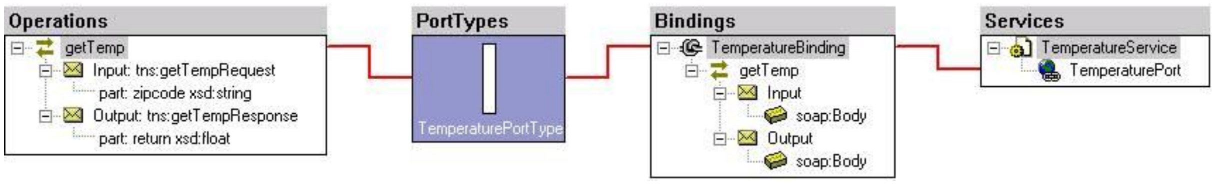


Figure 34. WSDL:XML Spy Diagram

This diagram will be used later in the thesis document to illustrate the proposed architecture. In the meantime, complete description of designed services can be found in the annex sections.

4.3.2.5 UNIVERSAL DESCRIPTION, DISCOVERY AND INTEGRATION (UDDI)

The Universal Description, Discovery and Integration (UDDI) specification is another major part of the Web Service architecture. According to its name, the goal of UDDI is to provide a system, which would enable users to find and publish Web Services [49].

It should be mentioned, that the term UDDI is actually used in two ways, i.e. specification and implementation. Firstly, UDDI describes an XML-based format to store data and an API to search and publish this data. Secondly, the UDDI (Business registry) is an existing implementation of the UDDI specification, which was launched by IBM and Microsoft in 2001. The following description refers to the UDDI as a technical specification.

The UDDI consists of three parts:

- UDDI cloud service to provide synchronised implementations of the UDDI;
- UDDI data model to describe companies and web services;
- UDDI API to search and publish data.

UDDI cloud service: Searching existing web services requires registries to store information about them. Although everyone can create his own, standalone registry, it would be easier to use a centralised registry rather than perform searching in several storages for a particular web service.

Therefore, UDDI cloud serves to provide a logically uniform, but physically distributed registry, consisting of different nodes, which are synchronized automatically.

Hence, Microsoft and IBM promote UDDI Business registry in Version 1, while major companies like HP, SAP or NTT prefer Version 2 of the same specification.

UDDI data model: This model determines a standard form to store and search for the information. It is defined by an XML schema (http://uddi.org/schema/uddi_v3.xsd), which describes the four main types of information:

- The business entity element includes some predominant information like its name, address and most importantly a unique businessKey for a business entity structure. Detailed information about services may be provided in the businessServices element, which is a container for businessService structures;
- The businessService element contains information about a web service, like its name, description or taxonomy codes. Each business service must have a unique identifier called serviceKey. If a business service is not embedded inside a business entity, it must refer to the businessKey of a business entity. In addition, the

business service contains a bindingTemplates structure, i.e. a container for zero or more bindingTemplate elements;

- The bindingTemplate provides a more technical view of the service. Beside other information, it describes an accessPoint of a service, i.e. an entry point of a web service. Depending on the service, this could be an URL, an email address or even a telephone number. Technical details are provided in the element called tModelInstanceDetails, which is a simplified container for the tModel;
- The tModel structure represents a technical specification. Although the requirements are very flexible (the tModel only must have a unique key, i.e. a name, and can point to any kind of descriptions), it contains mostly a hyperlink to a WSDL.

UDDI API: In the previous stage, the UDDI data model achieved describing of a web service in a meaningful and systematic way, which is prerequisite for a successful searching. Having described a web service, the major task is to provide this information to a consumer that should be capable to search for it. Therefore, the **UDDI API** has two functional parts, i.e. one for searching and requiring and the other for publishing.

The API functions are accessible via SOAP over HTTP. Although an inquiry call does not require any authentication, publishing calls naturally ask for a user account, while being sent via SSL. A detailed overview of supported methods can be found in [50].

As it was shown above, UDDI is a key element of an efficient web service infrastructure. Its functionality is essential for web services, which interact automatically in a dynamically changing environment.

4.3.2.6 Business Process Execution Language for Web Services (BPEL4WS)

BPEL provides an XML notation and semantics for specifying a business process behaviour based on Web Services. A BPEL4WS process is defined in terms of its interactions with partners. A partner may provide services to the process, require services from the process, or participate in a two-way interaction with the process. Thus, BPEL orchestrates Web Services by specifying the order in which it is meaningful to call a collection of services, and by assigning responsibilities for each of the services to partners. One can use it to specify both the public interfaces for the partners and to describe the executable process.

BPEL4WS represents a convergence of the ideas of the XLANG (Microsoft) and WSFL (IBM) specifications. Both XLANG and WSFL are replaced by the BPEL4WS specification.

The first specification, v. 1.0, was published on 31 July 2002 by IBM, with the next version 1.1 published on 05 May 2003.

In contrast to its predecessors, i.e. XLANG and WSFL, which were not adopted, BPEL specification has very good perspectives to be accepted as a standard by the OASIS consortium.

There exist already a number of implementations including IDE for visual BPEL programming.

4.3.2.7 Products and Implementations

There are many Web Services toolkits available for different programming languages. The list below is a selection of existing toolkits, both commercial and open source products.

Product	**URL**	**SOAP Version**	**WSDL Support**	**Language**
GLUE	http://www.themindelectric.com	1.1	Yes	Java
SOAP::Lite	http://www.soaplite.com	1.1	Yes	Perl
nuSOAP	http://www.nusoap.org	1.1	Yes	PHP
IBM WSIF	http://www.alphaworks.ibm.com/tech/wsif	1.1	Yes	Java
axis	http://xml.apache.org/axis/	1.1	Yes	Java
MS .NET	http://www.gotdotnet.com	1.2	Yes	VB, C#, C++, Java
gSOAP	http://www.cs.fsu.edu/~engelen/soap.html	1.2	Yes	C/C++

Since the interoperability of these services is of a major concern to all the providers of WebSevice and SOAP implementations, they agreed to conduct regular interoperability tests. The results of these tests can be found in [51].

4.3.2.8 Web Services Security

The W3C Web Services Architecture Requirements outline the following six important security considerations for a comprehensive security framework:

1. **Authentication**, which guarantees that the service is accessible for anyone with a verified identity;
2. **Authorisation**, which guarantees that the authenticated person has the right to access the service or data;
3. **Confidentiality**, which guarantees that the data passed between a requester and a provider is protected from eavesdroppers;
4. **Integrity**, which offers that the message was not modified in its path from a requestor to a provider;
5. **Non-repudiation**, which guarantees that the sender of a message would not deny that he/she has sent it previously;
6. **Accessibility**, which ensures that the service is always accessible and that it is not affected by attacks, like denial-of-service (DoS), outside or inside of the system hosting the service.

Nowadays, Web services security can be achieved at the following two levels:

- **Security at the transport level.** Security at the transport level uses the inbuilt security features of transport technologies like HTTP Basic Authorization, HTTP with secure socket layer (HTTPs), IBM WebSphere MQSeries, etc. ;

- **Security at the SOAP or messaging level.** This level is now a subject to an extensive research with the specifications being developed by groups like W3C and OASIS. They work upon usage of digital signatures, certificates, etc., at the XML document level. In addition, their work concerns the way the standardised framework includes XML-formatted security data into SOAP messages.

There are a lot of security-related activities going on in various standards organisations. These are the most important ones:

- XML Digital Signature, a W3C/IETF activity hosted at the W3C. This group has recently successfully defined an XML digital signature specification;
- XML Encryption, a W3C activity. This group has also completed its deliverable, i.e. a definition of the way of encrypting (portions of) an XML document;
- XML Key Management, a W3C WG that is currently defining a specification, that would allow clients to obtain crypto key information (keys, certificates, etc.) and to perform key management such as initial registration, revocation, etc. The requirements document is in the last call;
- OASIS Security Services TC develops SAML (Security Authorization Markup Language). SAML is a framework for exchanging identification information; for example, a trusted third-party (such as a PKI CA or a network login server) could provide a signed set of assertions certifying someone’s identity. SAML is the basis of the Liberty Alliance providing a single sign-on facility;
- OASIS Access Control Markup Language TC promotes XACML, which is a framework for defining a set of privileges required to perform an operation, including identity information and external factors (e.g. an access policy or the time of the day). The XACML documents are also in the last call phase;
- OASIS Digital Signature Services TC is a new committee with the goal to define an interface for a signature generation and verification service;
- OASIS Web Services Security TC is currently working on the WS-Security document by IBM and Microsoft, which defines the way of signing of a SOAP message. The group also intends to build a foundation for higher-level security services, such as policy integration, automatic interoperability, etc…

In addition, the Web Services Interoperability Organisation works on a security profile to ensure basic interoperability among vendors. IBM and Microsoft, along with other partners, have issued a Security Roadmap [52] that includes seven new specifications.

Since the standards are mostly subject to a further development, the number of toolkits and libraries available for developers is limited at the moment, reality that is beginning to change. Major vendors such as IBM, Microsoft, Sun, and Verisign provide toolkits and products supporting new and emerging standards. A number of relevant Java Specification Requests (JSRs) were submitted to the Java Community Process (JCP) as well.

4.3.3 CONCLUSIONS

Taking into account the general requirements defined in the Introduction section, the Web Services provide functionality covering the issues mentioned above:

- programming language independency;
- operating system independency;
- transparency on network protocol level;
- support of divers communication mechanisms: synchronous – request/response and asynchronous – solicit response;
- available APIs and toolkits;
- automatic source code generation tools.

These characteristics are very interesting in the context of the supervision middleware. Therefore, it was decided to investigate the Web Service applicability for supervision systems. For this purpose, an attempt was made to develop an architecture based on Web Services.

The following section describes in detail the proposed concept including a component model and messaging protocol.

4.4 PROPOSED ARCHITECTURE

While the technologies like WBEM concentrated on an information model expressed in XML, this research focused on the transport and integration mechanisms. Having analysed the Web technologies, we developed an architecture, which describes a supervision system in terms of Web Services. Our hypothesis was that this approach would permit to build a highly portable, interoperable and integrable infrastructure.

The work included:

- Determining of a general architecture: basic components and interactions between them;
- Development of a communication protocol: messages and operations, message format and data encoding;
- Determining of a transport mechanism.

This section explains the proposed architecture in details combining two approaches: UML and Web Services. The UML class diagrams depict main components, their relations and interfaces. Meanwhile the Web Services approach describes the components as services, i.e. data types to define messages, operations that represent services as a collection of input and output messages and transport for message delivery.

Considering the transport mechanism, our choice of SOAP was driven by the integrability and interoperability requirements, since it is the most widely used protocol in the Web Services. Describing the interfaces in UML, we resorted to an RPC notation of the operations, which represents an interaction/operation as a call of a remote method.

The next section elaborates the Component Model of supervision systems, which was briefly described in the **Definitions** section.

4.4.1 COMPONENTS AND OPERATIONS

The basic components of the distributed supervision system were determined in the **Definitions** section. Beside the simple components (**Supervisor** and **Delegate**), a **Compound Agent** should be identified to cover the situation when an agent combines properties of both Supervisor and Delegate agents. In other words, a Compound agent serves as a Delegate (Supervised Entity) for one group of agents and represents a Supervisor for another group. For example, a Repository can be used for persistent storing of the monitoring data. This Repository agent implements the Supervisor paradigm for agents that store their monitored parameters in it. At the same time, it implements Delegate paradigm to provide remote supervisors with the monitoring history. Thus, the Supervisor “delegates” its authority to a Compound Agent to provide supervision functionalities on a required group of agents. Figure 35 illustrates the Compound Agent concept.

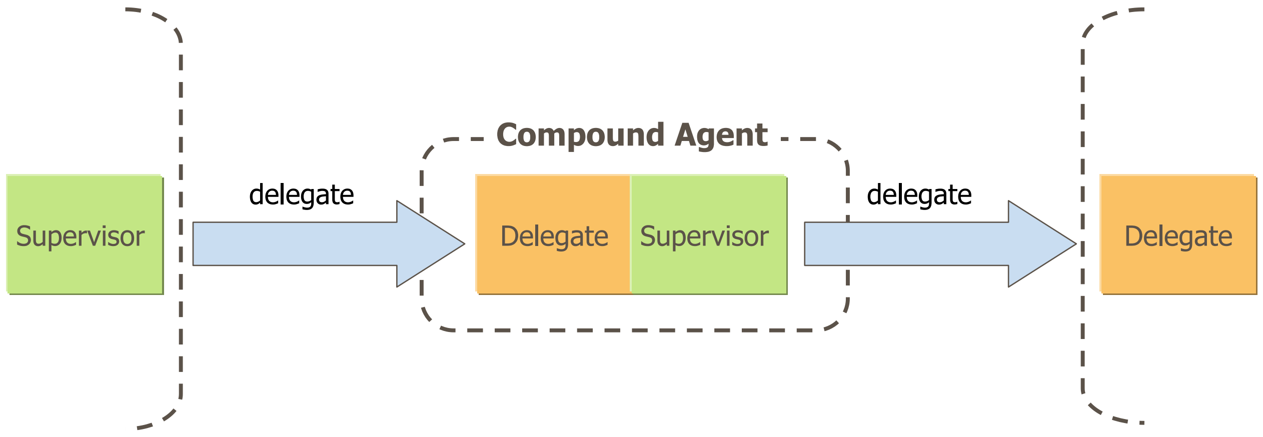


Figure 35. Compound Agent

In addition, working with multiple agents requires a discovery service. This service can be implemented by introducing a Directory Server – **Common Object Repository** (**CORE**) component. Hence, agents will store their capabilities, location and properties in a central site. For this purpose, additional operations are required:

- Register agent information;
- Discover an agent by a reference (location, functionality provided, etc.).

On the basis of these characteristics, relations between the component interfaces are identified, as depicted in Figure 36.

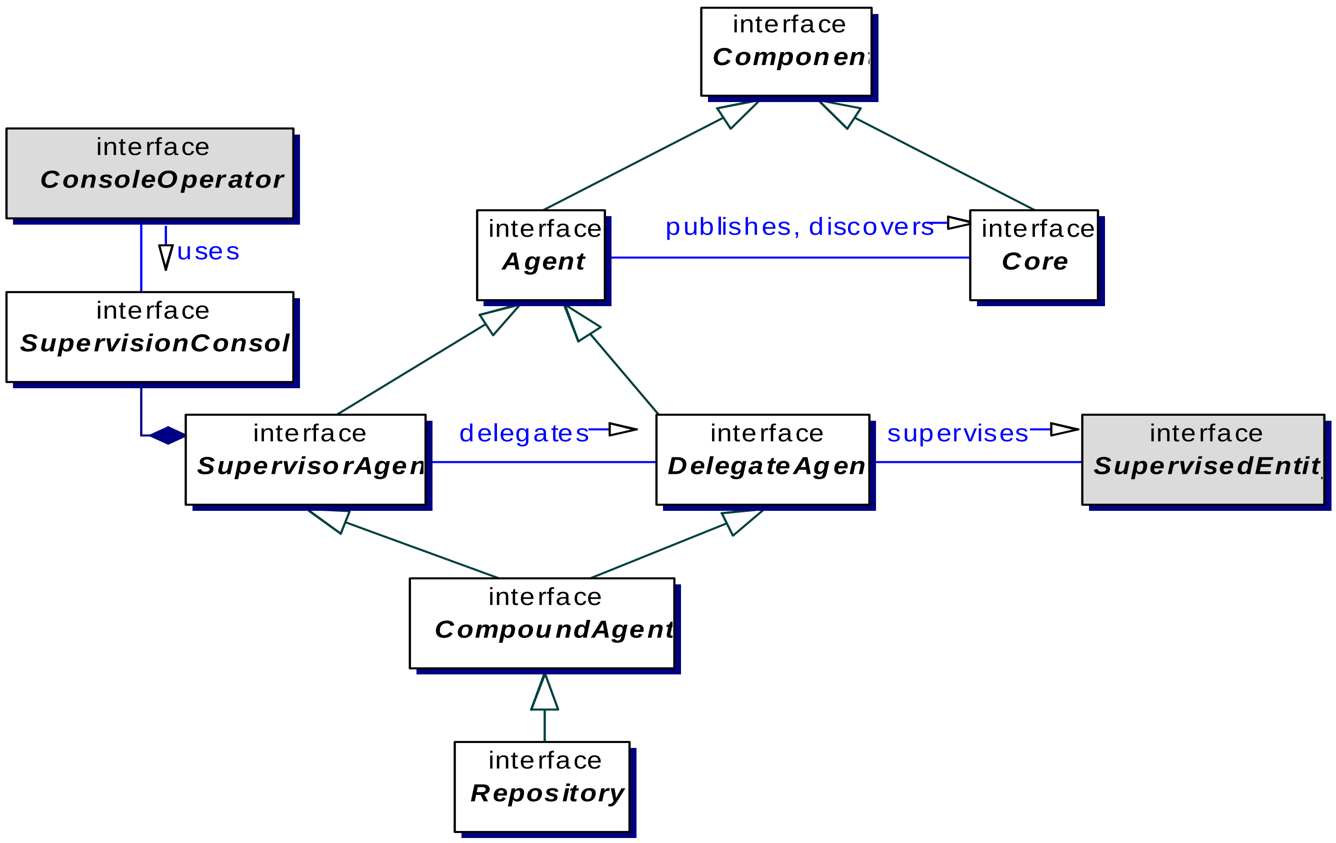


Figure 36. Component Interface Relations – UML Class Diagram

According to this diagram, Agent and Core classes inherit Component interface, in their turn, Supervisor and Delegate classes inherit Agent interface. Compound Agent class, as it was mentioned before, inherits both Supervisor and Delegate interfaces. Besides, Console Operator, i.e. administrator, uses Supervision console plugged to a Supervisor Agent. "Supervise" operations are defined depending on the interface of a Supervised Entity.

As for "delegate" operations, according to the requirements and the state of the art, one can point out that a distributed supervision system should support the following operations between a Supervisor and a Delegate:

- Query for supervision information;
- Request for command execution;
- Subscribe for an event;
- Response containing supervision information or a command execution return code;
- Response on an event.

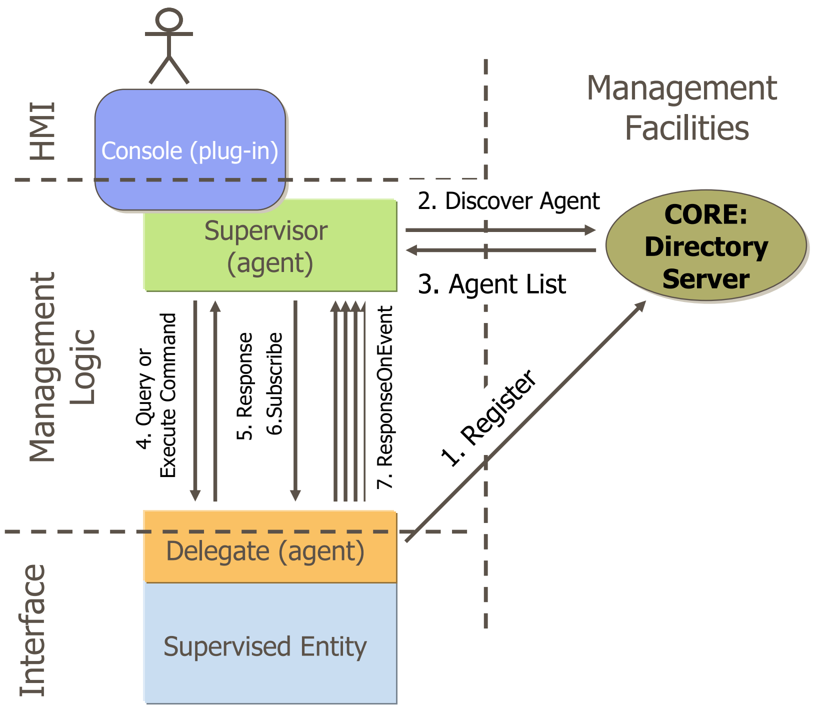


Figure 37. Agents' Life-Cycle.

Taking into account all the operations, an agents' life-cycle (Figure 37) is presented as follows:

- Initialisation (1,2,3 in Figure 37):
    - Delegate registers its properties on the CORE;
    - Supervisor discovers a list of agents corresponding to its needs.
- Functioning(4,5,6,7 in Figure 37):
    - Query for information:
        - Supervisor queries a particular agent for supervision information;

    - Delegate returns supervised parameters values.
  - Subscribe for information:
    - Supervisor subscribes for an agent event (status change, out of boundaries, etc.);
    - On event, Delegate returns supervised parameters values.
  - Perform action:
    - Supervisor requests Delegate for a command execution;
    - Delegate returns the acquired code after the command execution.

The next section thoroughly describes component interactions.

4.4.2 COMPONENT INTERACTIONS

Initialisation and functioning interactions mentioned above are described hereafter in detail.

For the initialisation process, the Core supports several operations. Three of them, **register**, **update registration** and **unregister**, are dedicated to keep the agent information up to date in Directory Server. Meanwhile, the search operation is purposed for an agent discovery.

The following two figures (Figure 38 and Figure 39) show sequences of initialisation operations.

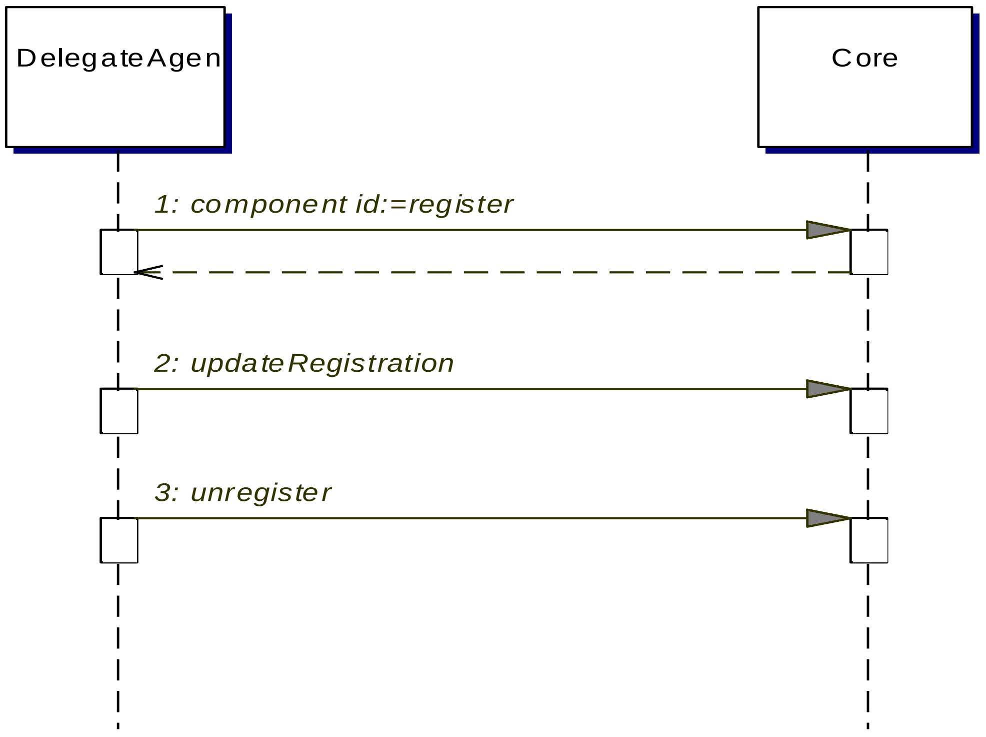


Figure 38. Delegate-Core Interactions – UML Sequence Diagram

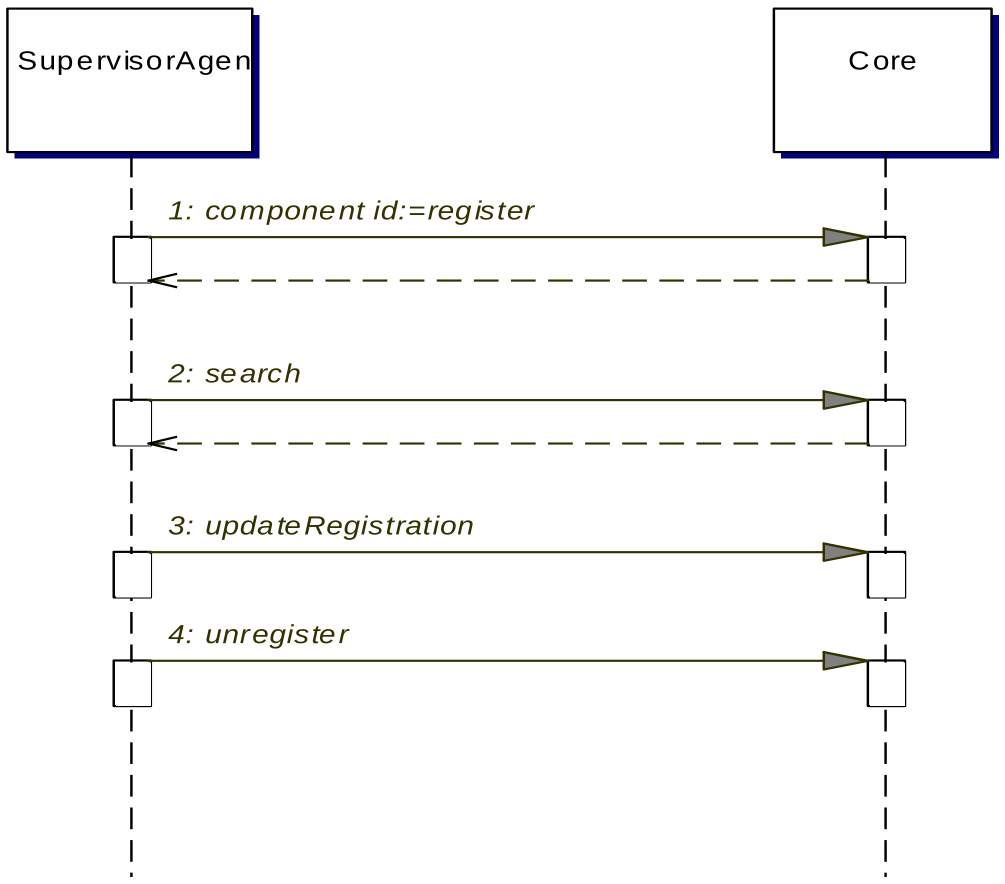


Figure 39. Supervisor-Core Interactions – UML Sequence Diagram

During the first operation, i.e. "**register"**, agents send their identification information to the Core and receive a unique Id. The identification information consists of:

- **Agent location**: it is an agent URL and, possibly, a place of the agent in a directory, i.e. custom information, like business unit or department this agent belongs to;
- **Agent type**: Delegate, Supervisor, Compound Agent;
- **Supported supervision operations**:
  - **Query/Response**: This field defines a format of query and response messages for each monitored parameter supporting this operation;
  - **Subscribe for an event**: This field defines a format of subscribe and "response on event" messages for each monitored parameter supporting this operation;
  - **Perform**: This field defines a format of the request for command execution, as well as format of the response containing command return code.
- **Custom identification information**: This section can contain some custom information, like agent dependencies, periodic events intervals, etc.

On **"update registration"** request, an agent may update its identification information, leaving its id, assigned by Core, unchanged. The **"unregister"** request removes the agent information from the Directory Server.

As for agent functioning, Figure 40 and Figure 41 depict the supervision operation interactions.

All interactions start with agents "handshake" (steps 1-2 in both figures), when they identify each other by the Agent Information mentioned above. On this step, agents verify their security certificates to decide whether to grant an access to their capabilities. It is an administrator's responsibility to set-up proper security policies for the supervision system components. After this, acquisition of monitoring data and command perform processes can be started.

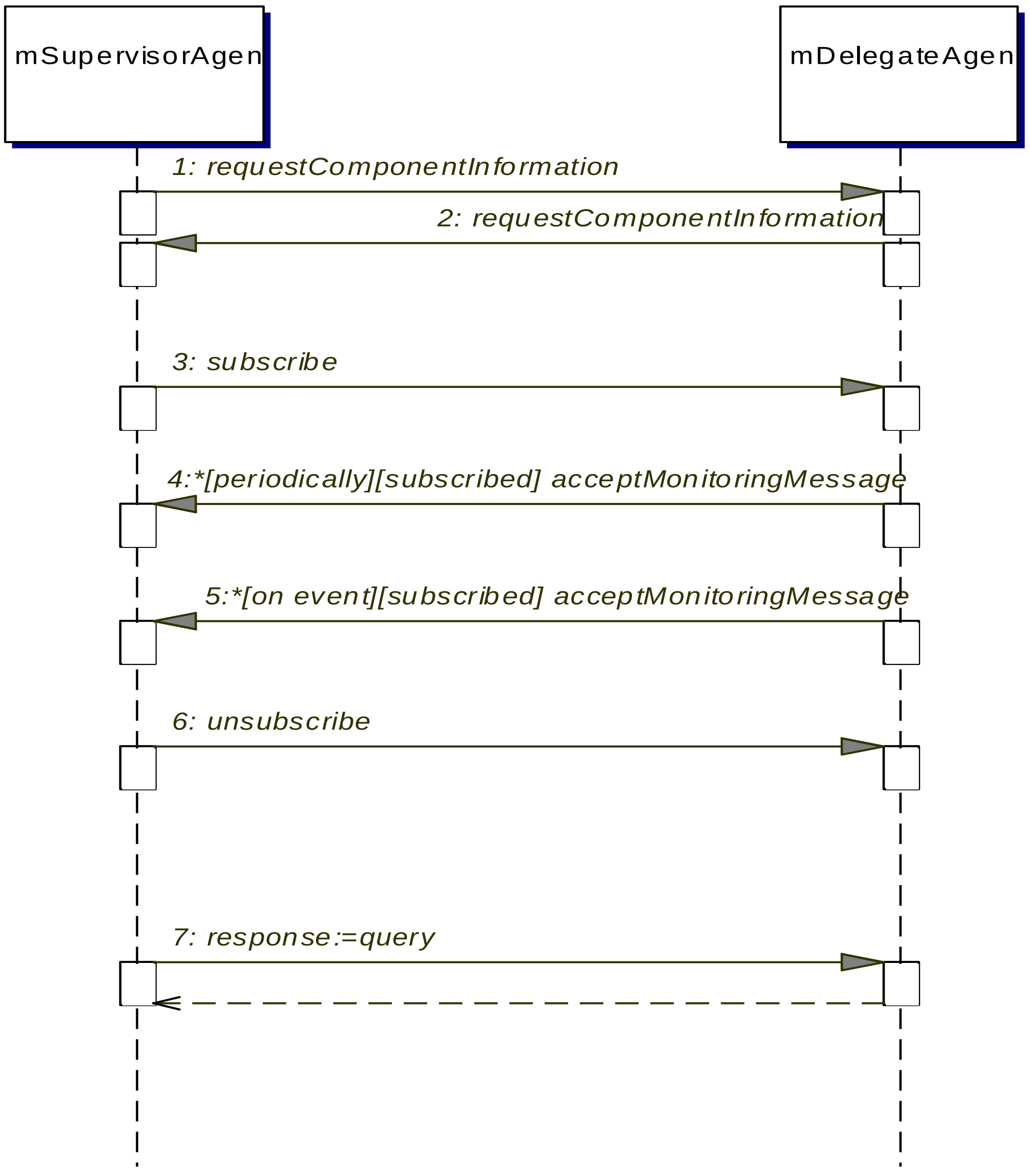

mSupervisorAgen
mDelegateAgen
1: requestComponentInformation
2: requestComponentInformation
3: subscribe
4:*[periodically][subscribed] acceptMonitoringMessage
5:*[on event][subscribed] acceptMonitoringMessage
6: unsubscribe
7: response:=query

Figure 40. Supervisor-Delegate Interactions – UML Sequence Diagram

Monitoring data acquisition can be achieved in a synchronous or an asynchronous manner. A Supervisor can query (Figure 40 - 7) for necessary monitoring information. A Delegate checks the monitored parameter right on request and returns the information to the Supervisor.

"On subscription" interaction starts with a subscription request (Figure 40 - 3). Then periodically or upon event, the Delegate sends monitoring information.

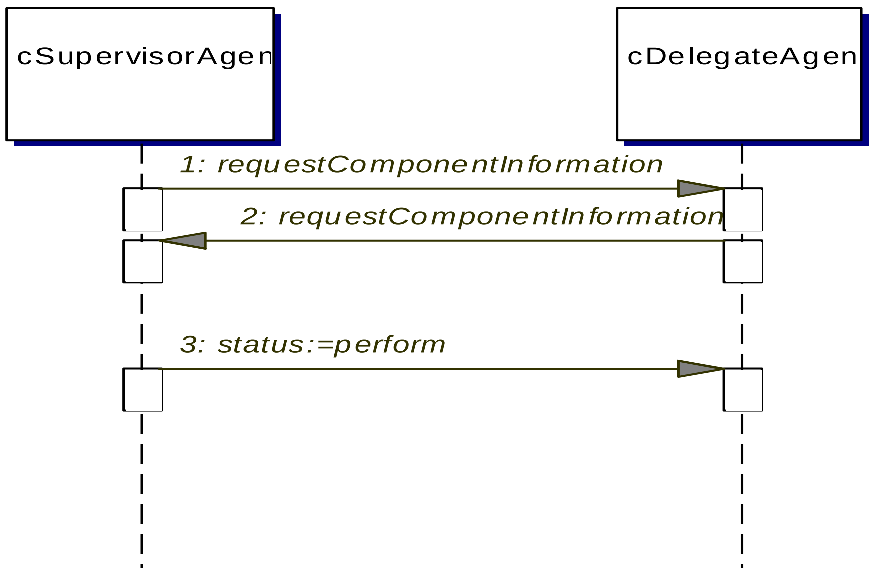

cSupervisorAgen
cDelegateAgen
1: requestComponentInformation
2: requestComponentInformation
3: status:=perform

Figure 41. Supervisor-Delegate Perform Interactions – UML Sequence Diagram

In the meantime, the "**perform**" interaction occurs always in synchronous manner. A Supervisor requests a Delegate for command execution and waits for a command return code.

The following section defines the component interfaces securing the described operations in case this architecture is mapped to Web Services.

4.4.3 COMPONENT INTERFACES

Web Services concept assumes that each communication operation is supported by the corresponding Service, which is a remote method (SOAP-RPC terminology [47]). The following diagram (Figure 42) illustrates methods that secure operations for each component of the architecture.

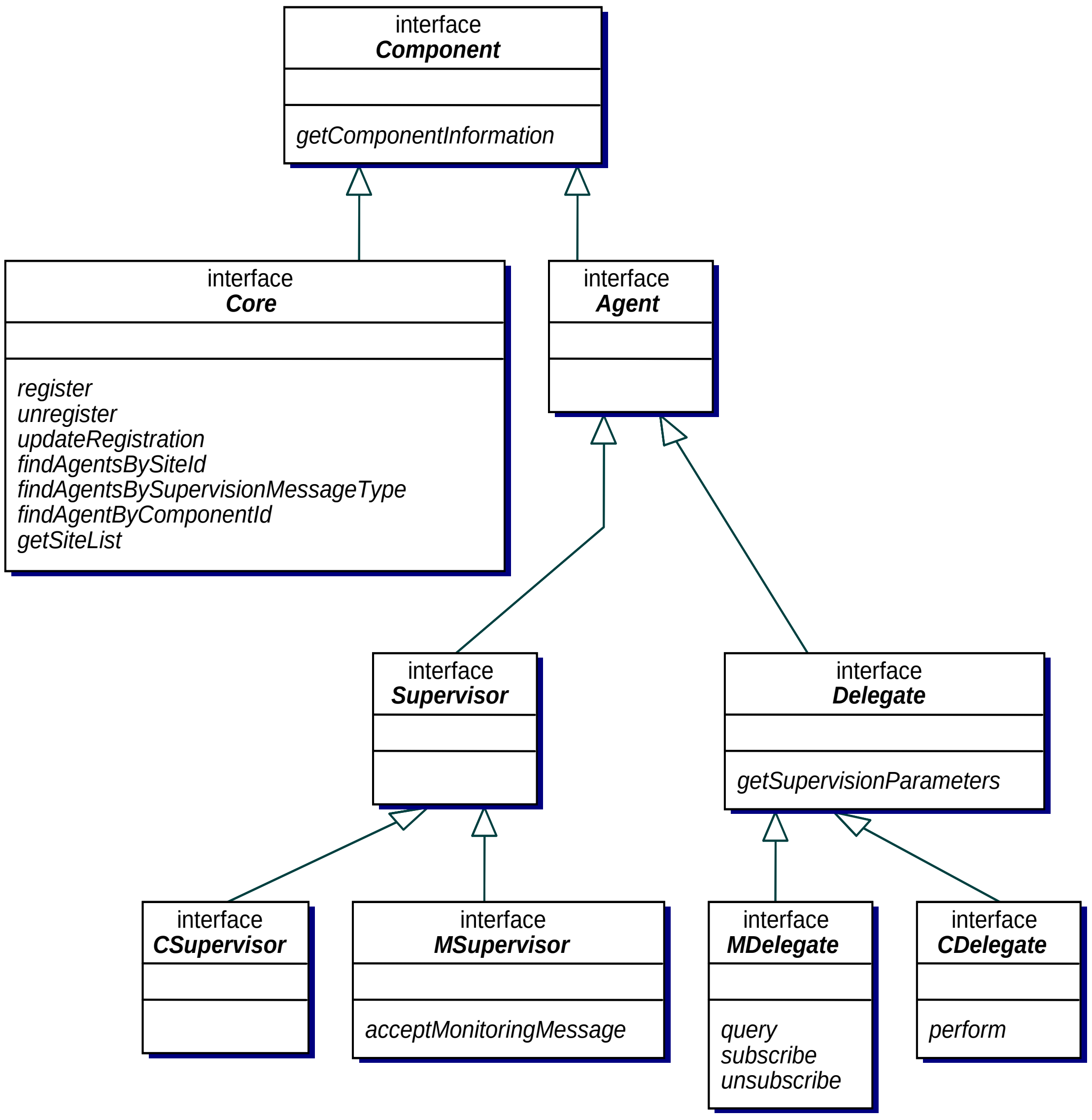


Figure 42. Component Hierarchy and Interfaces – UML Class Diagram

This diagram shows that the Core and the Agent inherit a Component interface, which has the only one method **getComponentInformation**. This method is used for mutual identification of components, for example, in a "handshake" operation. The Core has several methods for storing agent information and agent discovery. Registration operation is secured by **register**, **unregister**, and **update registration** methods described earlier. In the meantime, agent discovery operation is represented by several **find** methods. These methods allow searching for agent information stored in the Core. This information corresponds to identification information registered by agents themselves (will be addressed later).

The Supervisor and the Delegate extend the Agent interface. While the Supervisor has no general methods, the Delegate has **getSupervisionParameters** method, which is purposed to return types of supported supervision operations. This method can be used by a Supervisor to discover Delegate's capabilities.

The Control, Monitor Supervisor and Delegate (CSupervisor, MSupervisor, MDelegate and CDelegate) are subclasses of the Supervisor and the Delegate, which support control and monitor operations. The Monitor Supervisor supports **acceptMonitoringMessage** method, which is required for asynchronous communication, in case a Delegate responds eventually to a Supervisor. The Monitoring Delegate has several methods required for monitoring operations. The **query** method is used for a monitoring information request, while the **subscribe/unsubscribe** methods are used to subscribe to or resign from a given event.

The Control Supervisor is always originator of the request for a command execution. Hence, it has no methods for the control operation. Contrarily, the Control Delegate, as a service provider, has the **perform** method.

These methods are a transport mechanism for a messaging protocol. Incoming messages are passed as parameters of the methods, while response messages are passed as return values of these methods.

The following section describes the messaging protocol.

4.4.4 MESSAGES

A basic data communication unit is a Message. In this architecture, messages are classified in two types: supervision and service messages. The Message hierarchy is depicted in Figure 43 below.

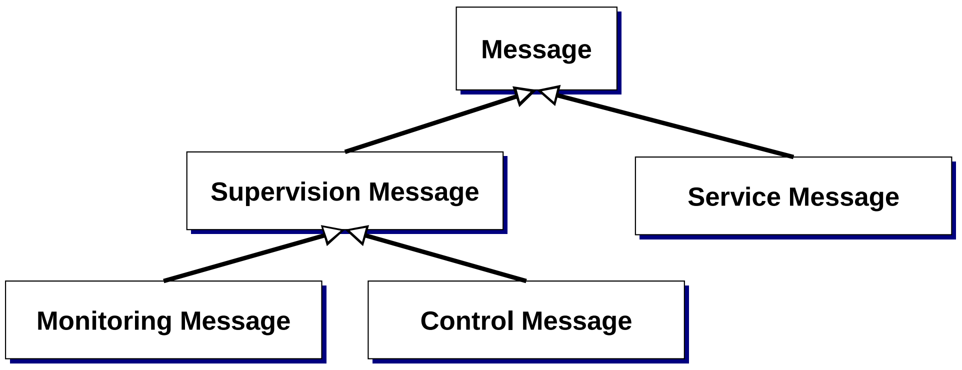


Figure 43. Message Hierarchy – UML Class Diagram

Supervision messages are data unites involved into monitoring data acquisition and into requests for execution operations. Accordingly, monitoring and control messages are intended to represent monitoring and control functions. Thus, these messages are involved only into communication between a Supervisor and a Delegate for the following operations:

- Monitoring:
    - Query/Response;
    - Subscribe/Response on Event.
- Control:
    - Perform Command execution.

A distinctive feature of these messages is that their contents are customised by developers to meet specific supervision goals.

The Service messages are used for “service” operations, a group of operations required for agent identification, discovery and “handshake”, as follows hereafter:

- Registration;
- Agent discovery;
- Component information exchange.

In addition, **Service Fault** messages belong to the type of service ones. These messages are generated during communication in the result of a system error. Service Faults indicate that the incoming message could not be processed due to some reasons, which are identified inside of the message (i.e. malformed message, database error, etc). Service Faults are defined for each operation and discussed later.

The service operations can be performed automatically, since Component Identification information is defined similarly for all kinds of agents. Although the message format is fixed, the encoding supports additional fields to allow message customisation.

The message structure is encoded using XML schema, that allows to validate resulting XML document (message) using its schema. This topic will be addressed later in the next chapter. XML does not impose specific requirements on the message format. However, a common practice is to divide message content into Service Information (Header) and Payload (Body) blocks. This structure is presented below in Figure 44.

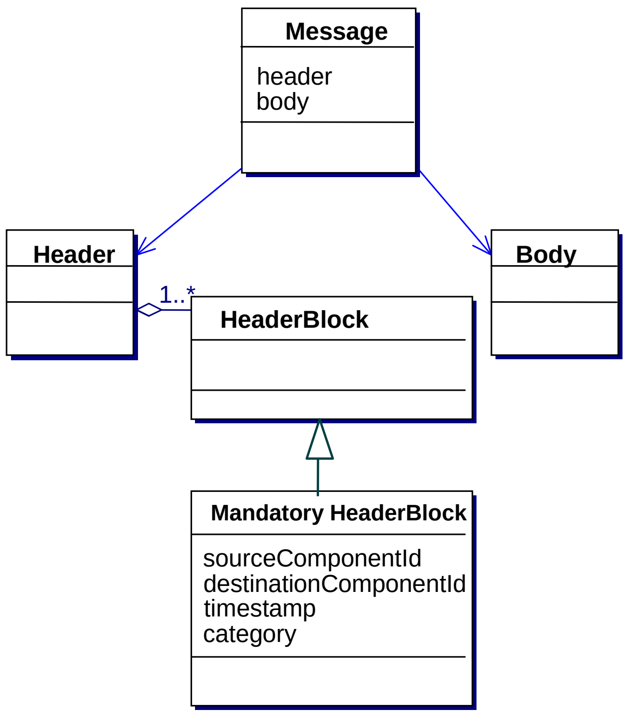


Figure 44. Message Structure – UML Class Diagram

The message can contain multiple header blocks including the mandatory one. The mandatory block consists of a small number of fields required for message identification, while additional headers can contain custom identification information like security signatures or others.

The minimum message identification information contains the following fields:

- **sourceComponentId**: A unique ID of the component from which the message originated;
- **destinationComponentId**: A unique ID of the component for which the message is destined;

- **timestamp**: A timestamp of message generation in UTC format;
- **category**: A message category that can be “monitoring”, “control” or “service”.

The Body block differs depending on the message category.

As depicted in Figure 45, the Supervision Message Body contains supervision information type and payload fields. These fields are fully customisable and are dedicated to reflect user needs concerning transition of some specific information. Payload type is defined by an agent developer, as well as a payload block.

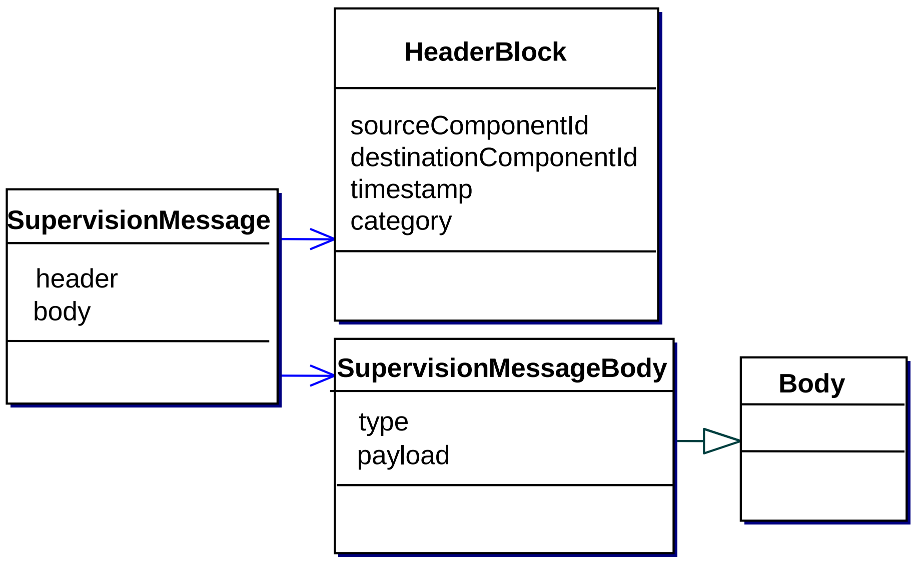


Figure 45. Supervision Message Structure – UML Class Diagram

The Payload type is defined by an agent developer, as well as a payload block structure. Although developers are invited to use XML schema for blocks definition, these blocks may be fully customised, according to developers’ needs and can contain any binary data. In this case, binary data handlers should be developed for both Supervisor and Delegate sides.

Service message structure, depicted in Figure 46, is similar to the one of a Supervision message.

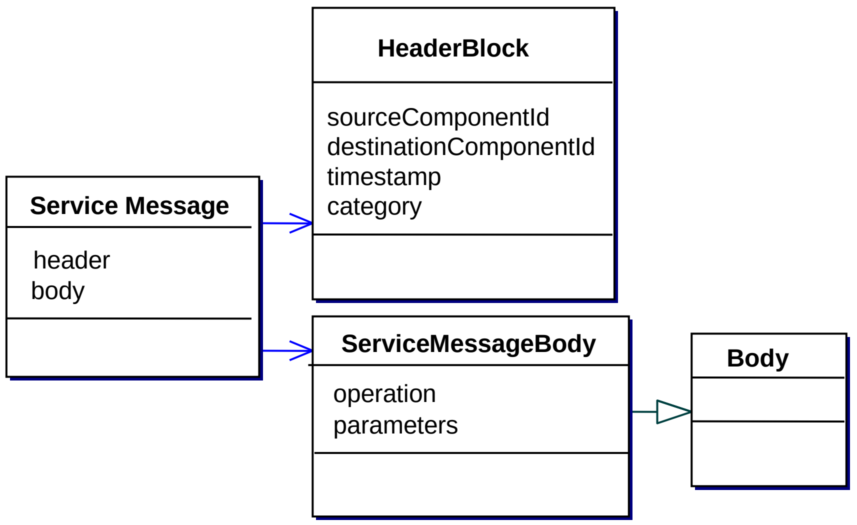


Figure 46. Service Message Structure – UML Class Diagram

The Service Message Body content is divided into two parts: Operation type and Parameters to be passed. Operation types have been described above, while Parameters for each operation will be outlined in the next section.

4.4.5 OPERATIONS – SERVICES ENCODING

The architecture supports various encoding types. In order to build a supervision framework for industrial application supervision (see Introduction), an XML based encoding has been developed to represent the above-mentioned system operations and messages.

The idea of this thesis is to map the proposed architecture to the Web Services concept. In this case, all the described operations should be represented as services. Therefore, the operation definitions should be encoded using Web Service Description Language (WSDL) [48], that is a standard mechanism to describe a service interface. This description can be helpful in automatic generation of an agent source code.

The operations, as well as service methods, were outlined in the previous sections. The following sections are purposed to define in detail the involved operations by means of WSDL.

Each service is bound to a corresponding operation and a transport mechanism. In the present encoding, it was chosen to use **rpc** (SOAP-RPC) in the function of a transport mechanism. The SOAP-RPC is the most widely supported communication model in Web Services.

The operations are defined by their input, output and fault messages. This data will be described later, after presentation of services.

4.4.5.1 CORE SERVICES

As it was described earlier, the Core provides the following services, depicted in Figure 47:

- register;
- unregister;
- updateRegistration;
- getSiteList;
- find.

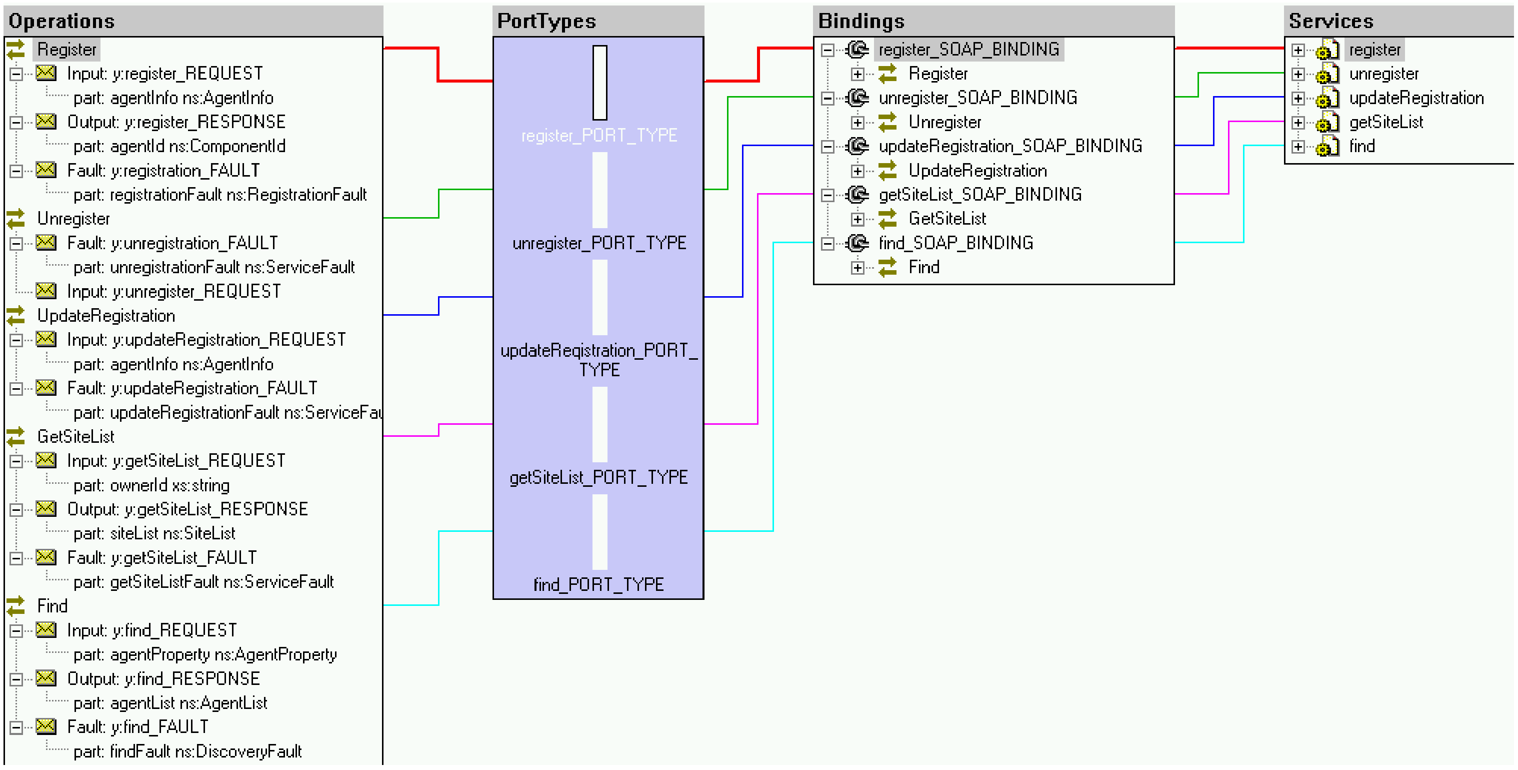

Operations
Register
Input: y:register_REQUEST
part: agentInfo ns:AgentInfo
Output: y:register_RESPONSE
part: agentId ns:ComponentId
Fault: y:registration_FAULT
part: registrationFault ns:RegistrationFault
Unregister
Fault: y:unregistration_FAULT
part: unregistrationFault ns:ServiceFault
Input: y:unregister_REQUEST
UpdateRegistration
Input: y:updateRegistration_REQUEST
part: agentInfo ns:AgentInfo
Fault: y:updateRegistration_FAULT
part: updateRegistrationFault ns:ServiceFa
GetSiteList
Input: y:getSiteList_REQUEST
part: ownerId xs:string
Output: y:getSiteList_RESPONSE
part: siteList ns:SiteList
Fault: y:getSiteList_FAULT
part: getSiteListFault ns:ServiceFault
Find
Input: y:find_REQUEST
part: agentProperty ns:AgentProperty
Output: y:find_RESPONSE
part: agentList ns:AgentList
Fault: y:find_FAULT
part: findFault ns:DiscoveryFault
PortTypes
register_PORT_TYPE
unregister_PORT_TYPE
updateRegistration_PORT_TYPE
getSiteList_PORT_TYPE
find_PORT_TYPE
Bindings
register_SOAP_BINDING
Register
unregister_SOAP_BINDING
Unregister
updateRegistration_SOAP_BINDING
UpdateRegistration
getSiteList_SOAP_BINDING
GetSiteList
find_SOAP_BINDING
Find
Services
register
unregister
updateRegistration
getSiteList
find

Figure 47. Core Services Definition – XMLSpy WSDL Diagram

“Register” operation takes agent information as a parameter and returns an agent identifier. In case of an error, it returns a Registration Fault.

“Unregister” operation requires an agent identifier that allows the Core to release a database entry.

“UpdateRegistration” operation deals with agent information entry to update the Core.

“GetSiteList” operation is specific for the chosen architecture implementation. Agent Identifier (described later) is a compound value, containing identifier of a “site” and “owner”. These characteristics are logical parameters to set up a hierarchy between agents. “GetSiteList” operation returns a list of “sites” for a given “owner”.

“Find” operation is purposed to locate agents specified by a set of properties (AgentProperty). These properties are part of the agent information definition.

For detailed information please refer to the annex section 9.1.

4.4.5.2 Delegate Services

Figure 48 shows a general case of a Delegate.

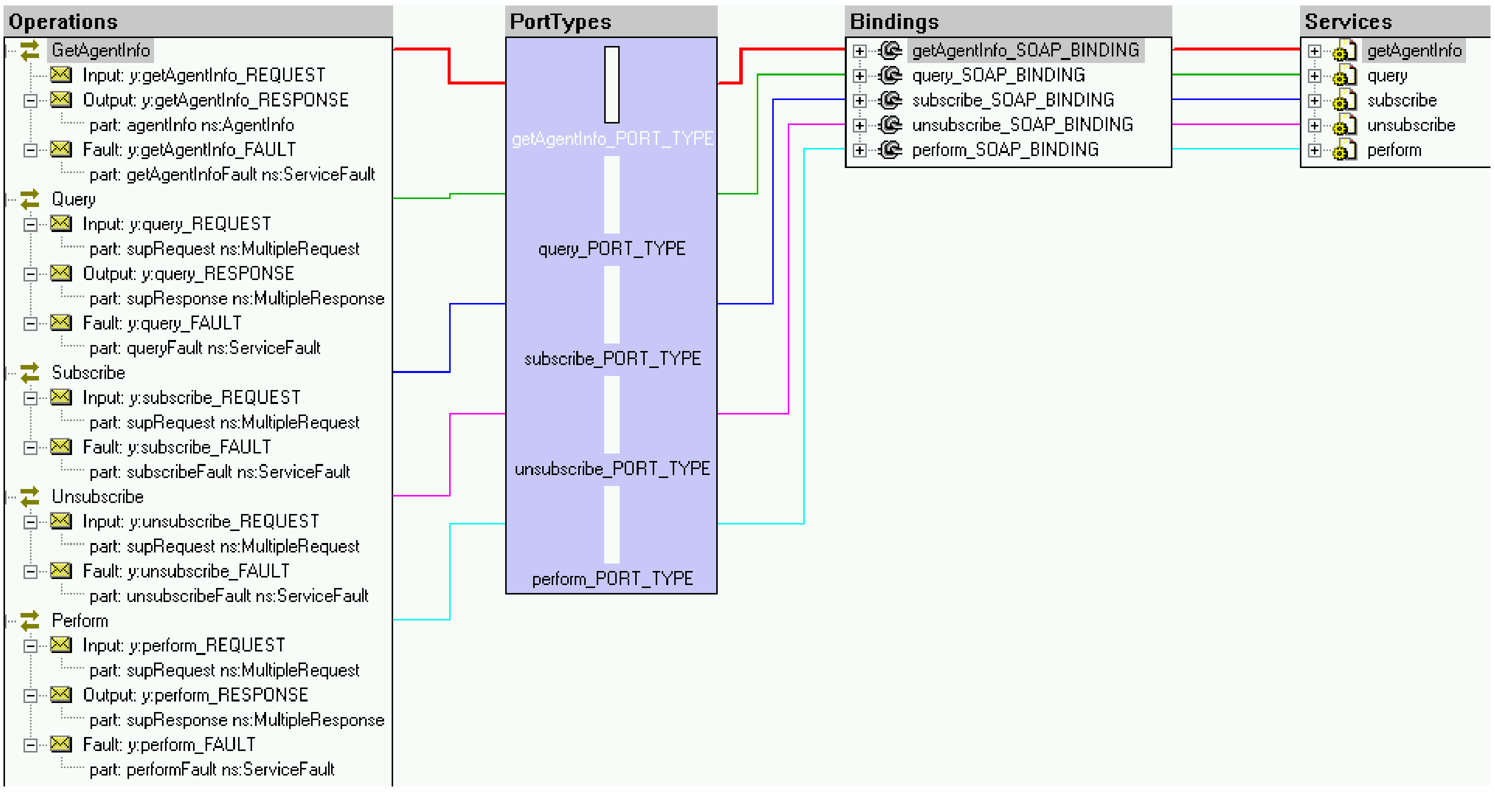

Operations
GetAgentInfo
Input: y:getAgentInfo_REQUEST
Output: y:getAgentInfo_RESPONSE
part: agentInfo ns:AgentInfo
Fault: y:getAgentInfo_FAULT
part: getAgentInfoFault ns:ServiceFault
Query
Input: y:query_REQUEST
part: supRequest ns:MultipleRequest
Output: y:query_RESPONSE
part: supResponse ns:MultipleResponse
Fault: y:query_FAULT
part: queryFault ns:ServiceFault
Subscribe
Input: y:subscribe_REQUEST
part: supRequest ns:MultipleRequest
Fault: y:subscribe_FAULT
part: subscribeFault ns:ServiceFault
Unsubscribe
Input: y:unsubscribe_REQUEST
part: supRequest ns:MultipleRequest
Fault: y:unsubscribe_FAULT
part: unsubscribeFault ns:ServiceFault
Perform
Input: y:perform_REQUEST
part: supRequest ns:MultipleRequest
Output: y:perform_RESPONSE
part: supResponse ns:MultipleResponse
Fault: y:perform_FAULT
part: performFault ns:ServiceFault
PortTypes
getAgentInfo_PORT_TYPE
query_PORT_TYPE
subscribe_PORT_TYPE
unsubscribe_PORT_TYPE
perform_PORT_TYPE
Bindings
getAgentInfo_SOAP_BINDING
query_SOAP_BINDING
subscribe_SOAP_BINDING
unsubscribe_SOAP_BINDING
perform_SOAP_BINDING
Services
getAgentInfo
query
subscribe
unsubscribe
perform

Figure 48. Delegate Services Definition – XMLSpy WSDL Diagram

The Delegate provides the following services:

- **getAgentInfo** service is used for an agent "handshake" mechanism, allowing the agents to discover each other;
- **query** secures synchronous communication model (Query/Response mechanism);
- **subscribe** provides a mechanism for a Supervisor to express its interest in a particular supervised parameter or event (part of an asynchronous communication model);
- **unsubscribe** allows a Supervisor to resign a subscription;
- **perform** represents a control mechanism, allowing a Supervisor to command a Delegate.

Communication messages contain the two main types of data: MultipleRequest and MultipleResponse. They represent a general form of a supervision parameter. That includes a capability to request and to pass a parameter with possibly multiple and nested entries. For example, in a request one can define a necessary parameter using several values: "Give me a disk load of the **disk C:** of **host XXX** of **network YYY**". This concept is described later in the Encoding section.

For further details concerning the Delegate services definition, please refer to the annex section 9.2.

4.4.5.3 SUPERVISOR SERVICES

The Supervisor supports only two services for communication with the environment (Figure 49):

- **getAgentInfo** provides the same information as described for a Delegate;
- **acceptMonitoringMessage** allows a Delegate to asynchronously return values to the Supervisor. This service requires "MultipleResponse" parameter, similar to the value returned by the Query operation of a Delegate.

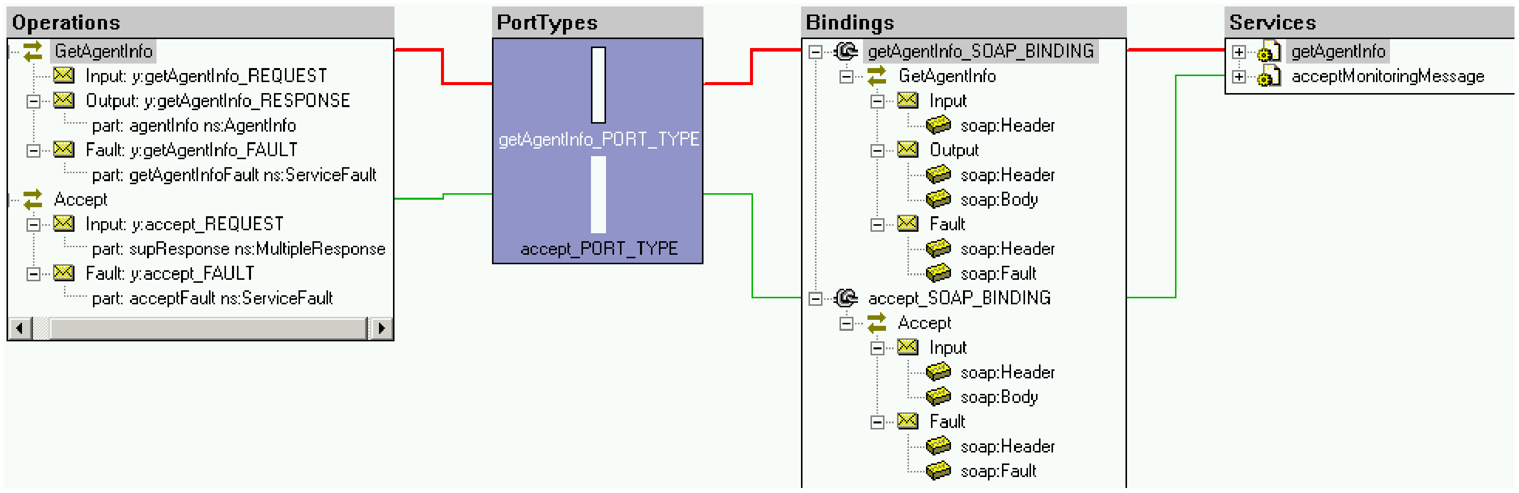


Figure 49. Supervisor Services Definition – XMLSpy WSDL Diagram

Detailed information about service definitions can be found in the annex, section 9.3.

The next section presents information that is exchanged between the components.

4.4.6 INFORMATION ENCODING

The messages are the only data entities exchanged in the supervision system. They are composed of the header and body blocks. The header block is purposed to identify the message. Although developers can customise this information, the MandatoryHeaderBlock is required by the supervision system. The body block differs depending to the type of a message: service or supervision. The service messages are fixed, while the supervision ones are defined in a general way. The supervision messages need to be customised by the developers.

In addition, the system data model can be easily changed overriding the basic types.

The message description starts with identification of the Mandatory Header Block.

4.4.6.1 Mandatory Header Block

As described earlier, each message contains the Mandatory Header Block. Its data fields are defined, as follows:

Name	*Type in XML schema*	*Description*
sourceComponentId	ComponentId	ID of the component from which the message originated
destinationComponentId	ComponentId	ID of the component for which the message is destined
timestamp	xsd:long	Timestamp of message generation in UTC format; number of milliseconds since January 1, 1970, 00:00:00 UTC
category	xsd:string	Message category: monitoring, control or service.

In XML Schema language, this block can be defined, as follows:

```
<xsd:complexType name="MandatoryHeaderBlock">
        <xsd:sequence>
            <xsd:element name="sourceComponentId" type="ComponentId"/>
            <xsd:element name="destinationComponentId" type="ComponentId"/>
            <xsd:element name="timestamp" type="xsd:long"/>
        </xsd:sequence>
 </xsd:complexType>
```

In this definition the ComponentId type represents a component identifier, which is described in the next section.

4.4.6.2 Component Identification

The Core and each agent have unique identifiers that are called Component Ids. This id is purposed to establish a logical hierarchy within the agent group. Besides, this id is used, as one of the criteria to locate a needed agent. It was decided to present the structure of this id, as follows:

<Component Id> = <Core Id>.<Owner Id>.<Site Id>.<Agent Id>

All ids are string values that uniquely identify some aspects of agent distribution.

Core Id identifies the Core (Directory Server) of the supervision system. In order to participate in particular supervision system installation, an Agent should be registered within the Core of this system. For that purpose, each Agent should be aware about the Core host address. Supervision System Administrator is supposed to set this address during the agent deployment. Default Core Id value is “0”, which is set on the agents that have never been registered (communicated with CORE).

Owner Id is intended to identify the owner of the component. The concrete semantics depends on a given deployment model. Usually, the owner will correspond to a business unit or a client in multi-client installations, like regions and nodes in the Tivoli framework. For example, IBM Corporation has several supervised sites (groups of resources), in this case Owner Id can be set to “IBM Corp”, etc. “0” value signifies that the owner Id is not applicable.

Site Id is allocated to each site, group of supervised resources linked logically (i.e. Oracle Server host, Distributed Simulation group of hosts). Based on a certain criteria, the components can be logically grouped into collections referred to as sites, e.g. collections of components belonging to the same security domain or being physically co-located. The semantics depends on a given supervision model and is responsibility of Supervision System Administrator. “0” value corresponds to the cases when site Id is not applicable (e.g. when single-site installation is chosen).

Agent Id is unique for each Agent, which is automatically generated by the Core when a component registers within the supervision system. “0” value corresponds to the cases when Agent Id is not registered on the Core.

Agents should use “0.0.0.0” as a component Id, in case of referencing the default core.

Using XML schema, component id is defined, as follows:

```
<xs:complexType name="ComponentId">
    <xs:sequence>
        <xs:element name="coreId" type="xsd:string"/>
        <xs:element name="ownerId" type="xsd:string"/>
        <xs:element name="siteId" type="xsd:string"/>
        <xs:element name="agentId" type="xsd:string"/>
    </xs:sequence>
</xs:complexType>
```

The next section describes another important data entity, i.e. information about an agent.

4.4.6.3 Agent Information

Defining the agent in WSDL is not explicit, since it defines the component in general, leaving developing of supervision operations to a user. In addition, supervision system functioning requires detailed information about agent and its capabilities. The supervisors should be able to locate the right agent. That is why, almost all service operations, including registration and discovery, require transmission of information specifying the agent. This information comprises the following agent characteristics:

- **Component Id** described earlier;
- **Location**: URL to the agent services;
- **Type**: Delegate or Supervisor;
- **Name**: for example, “Linux System Monitoring Agent”;
- **Description**: a textual information about the agent, for example, its limitations;
- Supported **Monitor** and **Control** operations. For example, they can be “Query/Response”, “Subscribe/ResponseOnEvent” or “Perform”;

- **Availability**: whether the agent is on and running;
- **Definition of the Monitoring/Control operations**: XML schema defining the Monitoring/Control operations, including **Interaction Models** (Query/Response, Subscribe/ResponseOnEvent, Perform) for each operation.

The following diagram illustrates agent information structure:

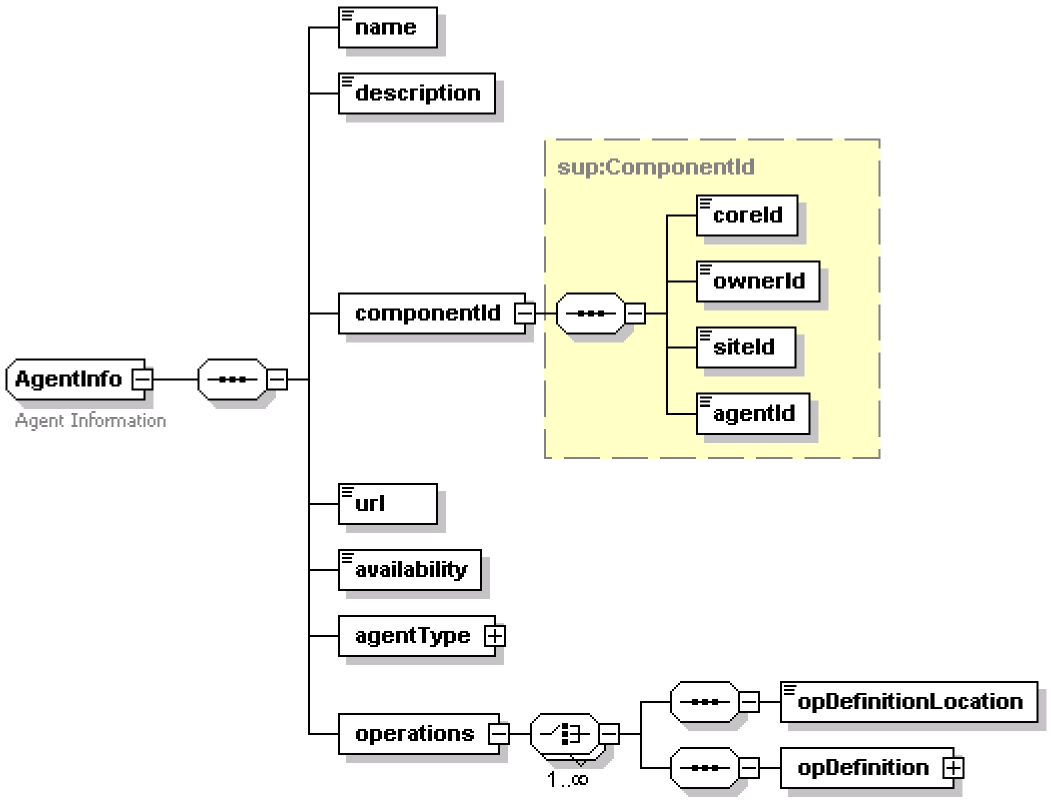


Figure 50. Agent Information Structure – XMLSpy XSD Diagram

In the form of an XML schema, this definition is presented as follows:

```
<xs:complexType name="AgentInfo">
    <xs:annotation>
        <xs:documentation>Agent Information</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="description" type="xs:string"/>
        <xs:element name="componentId" type="sup:ComponentId"/>
        <xs:element name="url" type="xs:anyURI"/>
        <xs:element name="availability" type="xs:boolean"/>
        <xs:element name="agentType" type="sup:AgentType"/>
        <xs:element name="operations">
            <xs:complexType>
                <xs:choice maxOccurs="unbounded">
                    <xs:sequence>
                        <xs:element name="opDefinitionLocation" type="xs:anyURI"/>
                    </xs:sequence>
                    <xs:sequence>
                        <xs:element name="opDefinition" type="sup:SupOperationDefinition"/>
                    </xs:sequence>
                </xs:choice>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
```

Where AgentType and operations are defined hereafter.

AgentType: Agent type can be both Supervisor and Delegate. Compound Agents possess these two types simultaneously. Therefore the schema is as follows:

```
<xs:complexType name="AgentType">
    <xs:annotation>
        <xs:documentation>Type of Agent</xs:documentation>
    </xs:annotation>
    <xs:choice>
        <xs:choice>
            <xs:element name="supervisor" type="SupervisorAgentType"/>
            <xs:element name="delegate" type="DelegateAgentType"/>
        </xs:choice>
        <xs:sequence>
            <xs:element name="supervisor" type="SupervisorAgentType"/>
            <xs:element name="delegate" type="DelegateAgentType"/>
        </xs:sequence>
    </xs:choice>
</xs:complexType>
```

Operations: Each agent should explicitly define its operations, which are services, in terms of Web Services. Therefore the operations are defined in wsdl format, and location of these wsdl descriptions is necessary for agent information:

```
<xs:element name="operations">
    <xs:complexType>
        <xs:choice maxOccurs="unbounded">
            <xs:sequence>
                <xs:element name="opDefinitionLocation" type="xs:anyURI"/>
            </xs:sequence>
            <xs:sequence>
                <xs:element name="opDefinition" type="sup:SupOperationDefinition"/>
            </xs:sequence>
        </xs:choice>
    </xs:complexType>
```

Example: The following example shows possible agent information in XML format:

```
<?xml version="1.0" encoding="UTF-8"?>
<agentInformation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="D:\andrey\thesis\concept.xsd">
    <name> System Agent for Free Disk Space Check</name>
    <description>System Agent for disk space monitoring</description>
    <componentId>
        <coreId>MainCore</coreId>
        <ownerId>EADS</ownerId>
        <siteId>DISdepartment</siteId>
        <agentId>SystemAgentOnKing</agentId>
    </componentId>
    <url>http://eads.com/SystemAgent</url>
    <availability>1</availability>
    <agentType>
        <supervisor>Supervisor</supervisor>
    </agentType>
    <operations>
        <definitionLocation>http://eads.com/SystemAgent/diskEx.xsd/opDef</definitionLocation>
    </operations>
</agentInformation>
```

A more detailed definition of the Agent Information structure (AgentInfo) can be found in the annex, section 9.4.

4.4.6.4 SUPERVISION OPERATION ENCODING

The Supervision operations include monitoring information acquisition and command execution. These operations are not defined by the WSDL and should be customised for each agent type. In a general form, the operation definition is depicted in Figure 51:

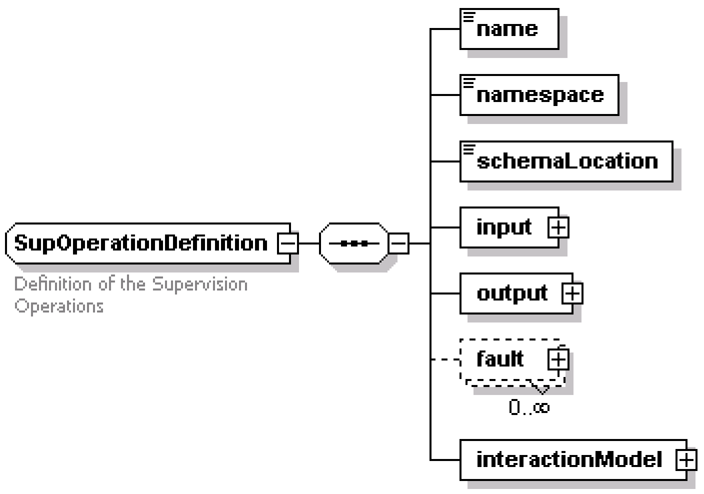


Figure 51. Supervision Operation Definition – XMLSpy XSD Diagram

Each supervision operation is characterised by its input/output parameters, operation faults and interaction models implemented for this operation. Although, all these parameters are defined by the developers of each particular agent, a generic form of request/response parameters is described in the nest section.

SchemaLocation field is used to indicate the schema, which describes input parameters, while “namespace” field is provided to eventually distinguish parameters with the same name.

In addition, developers can define a list of faults that are used to signal about an illegal state of an operation.

As it was mentioned earlier, the interaction models can be “Query/Response”, “Subscribe/ResponseOnEvent” or “Perform”. They define which agent method (Web Service) is used for operation processing. These methods may be “query”, “acceptMonitoringMessage” or “perform”.

More detailed information about the Supervision Operation Definition structure (SupOperationDefinition) can be found in the annex, section 9.4.

4.4.6.5 SUPERVISION REQUEST ENCODING

It is developers, who define details of requests for supervision operations. However, the following general form of a parameterised request can be useful as an example. The developers can simply override this definition according to their needs.

Figure 52 depicts a structure of a request.

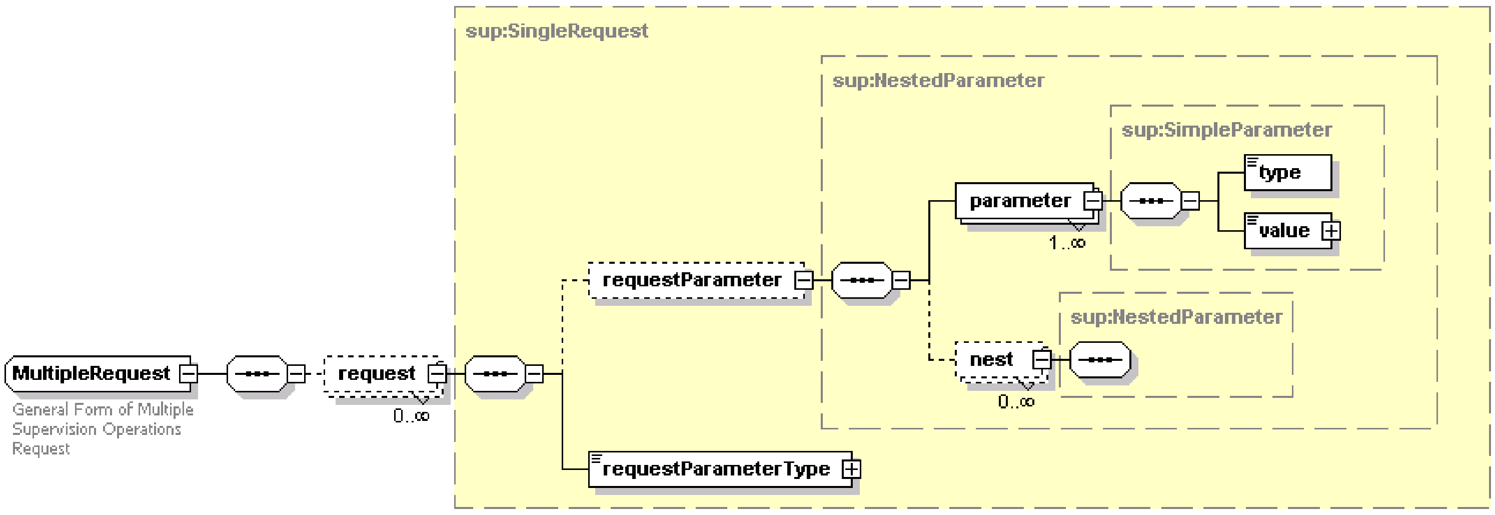


Figure 52. General Request Definition – XMLSpy XSD Diagram

A request contains a type of required information (requestParameterType), for example, amount of a free disk space. In addition, the information identifying the requested parameter should also be passed, for example, a name of the considered host and a name of the disk. In order to reflect the hierarchy in the request, a nested parameter concept is introduced here. Thus, all levels of information can be indicated in the request (i.e. Host : Disk : DiskFreeSpace).

The following diagram (Figure 53) represents a structure of a request for the "Disk Free Space" example.

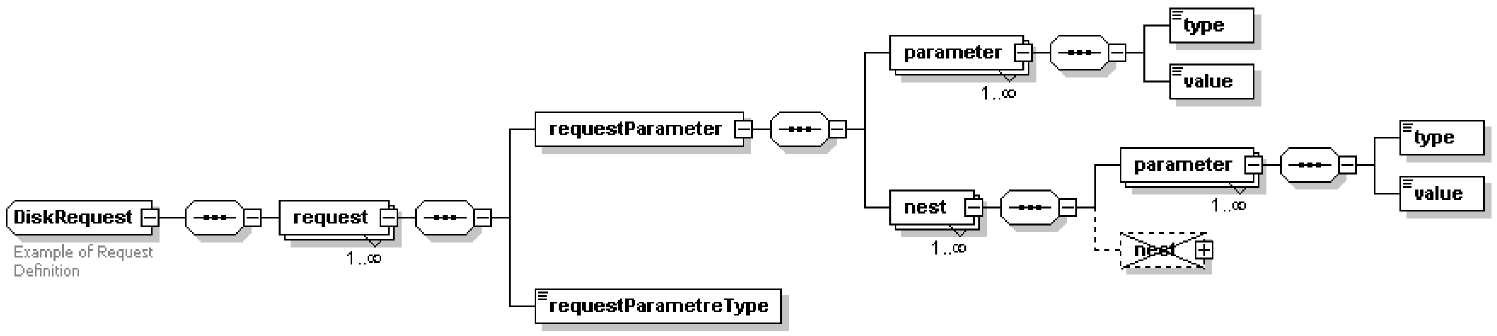


Figure 53. Request for Free Disk Space – XMLSpy XSD Diagram

The first parameter of the request is a Host Name, while the second, nested parameter contains Disk Name information.

In an XML form, this request can look, as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<diskReq xmlns="http://disk.example.namespace" xmlns:fault="http://fault.types.namespace"
xmlns:sup="http://sup.types.namespace" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://disk.example.namespace
D:\andrey\thesis\XSD\diskEx.xsd">
    <request>
        <requestParameter>
            <parameter>
                <type>sys:HostName</type>
                <value>King</value>
            </parameter>
            <nest>
```

```
                <parameter>
                    <type>sys:DiskName</type>
                    <value>C:</value>
                </parameter>
            </nest>
        </requestParameter>
        <requestParametreType>sys:FreeSpace</requestParametreType>
    </request>
</diskReq>
```

The nested parameters concept is similar to a mechanism of MIB access for the SNMP architecture. Therefore, this architecture can be easily mapped to the proposed one.

A more detailed definition of the General Request structure (MultipleRequest) can be found in the annex, section 9.4. For the definition of the “Disk Free Space” example, please refer to the section 9.5.

4.4.6.6 Supervision Response Encoding

The response in general form is similar to the request parameter, depicted in Figure 54.

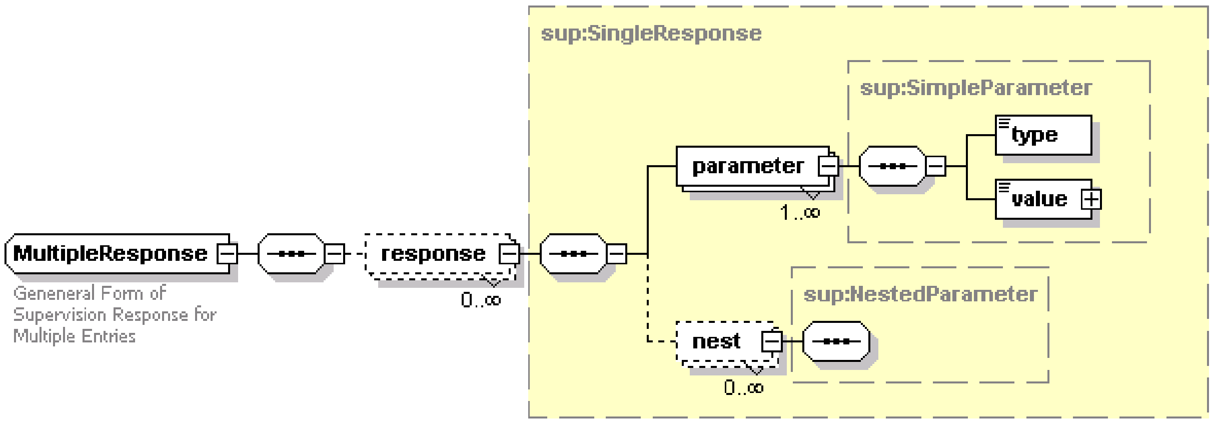


Figure 54. General Response Definition – XMLSpy XSD Diagram

Simple parameters, as well as nested ones can be passed as a response. The nested parameters can identify a hierarchy of retuned data. For the “Disk Free Space” example mentioned earlier, the nested parameters can be used to identify a particular disk characterised by its Host Name and Disk Name.

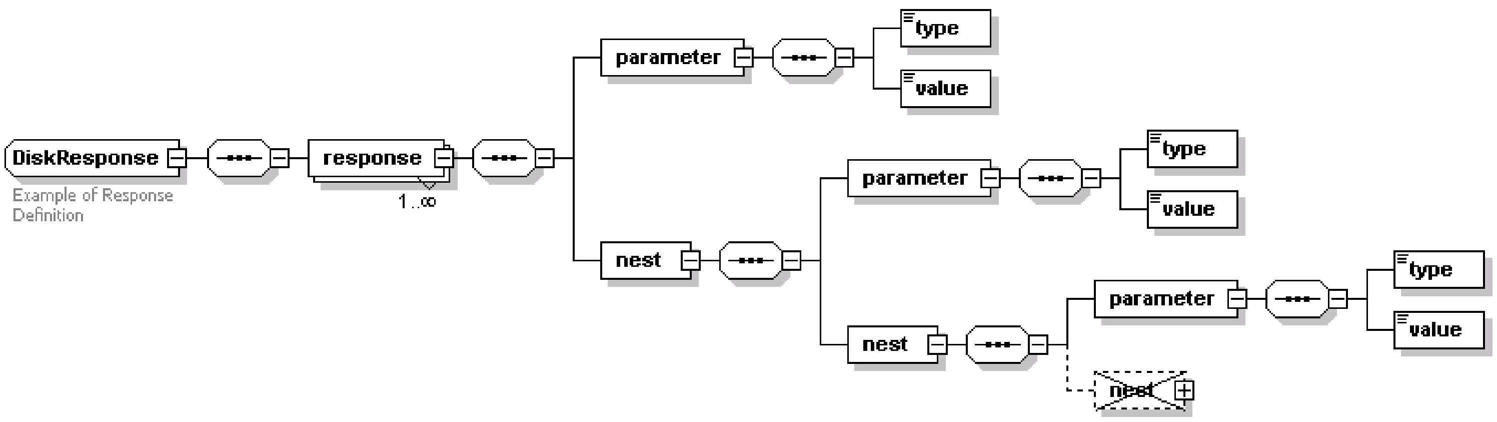


Figure 55. Response with Free Disk Space – XMLSpy XSD Diagram

In Figure 55, the first parameter is used for the HostName identification. The first nested parameters contain DiskName, while the lower parameter indicates an amount of a FreeSpace.

In an XML form a request can look, as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<diskRes xmlns="http://disk.example.namespace" xmlns:fault="http://fault.types.namespace"
xmlns:sup="http://sup.types.namespace" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://disk.example.namespace
D:\andrey\thesis\XSD\diskEx.xsd">
    <response>
        <parameter>
            <type>sys:HostName</type>
            <value>King</value>
        </parameter>
        <nest>
            <parameter>
                <type>sys:DiskName</type>
                <value>C:</value>
            </parameter>
            <nest>
                <parameter>
                    <type>sys:FreeSpace</type>
                    <value>1235</value>
                </parameter>
            </nest>
        </nest>
    </response>
</diskRes>
```

A more detailed definition of the General Response structure (MultipleResponse) can be found in the annex, section 9.4. For the definition of the “Disk Free Space” example, please refer to the section 9.5.

4.4.7 Conclusions

This section presented our approach for an architecture of a supervision framework. Our concern was about the communication transport and an integration mechanism providing for the best interoperability and flexibility. For this reason, we developed our solution upon Web-Services.

In order to explain the proposed architecture, we used UML diagrams to describe basic components, their relations and interactions. In the meantime, Web Service Definition Language served to transform the general architecture to Web Services.

It should be underlined, that we focused on communication keeping payload encoding out of the scope. Hence, for the Supervision Messages, the architecture only defines an envelope that then can encapsulate all kind of data according to developers’ supervision goals. However, for the flat monitoring information, the architecture proposes an encoding, which permits to incorporate various types of data including complex, compound and nested ones. This flexibility is illustrated with the “Disk Free Space” example.

The next section describes in detail a prototype developed to show concept applicability in real industrial systems, which were introduced in the section.

4.5 PROTOTYPE

The main goal of the prototyping is to build means purposed to validate and to evaluate the middleware concept in the context of the presented industrial problems (section 2.2). Therefore, specific supervision solutions were designed for these industrial systems.

This section describes the author's contributions in the frame of the GeneSyS project to the development of supervision solutions:

- Design of the Supervision Solutions for the Preliminary Design Review and the Distributed Training systems;
- Agent Collaboration for Generic Visualisation – a simple approach for description of flat supervision operations in order to build a generic console;
- Summarising View – a simple intelligent approach for information summarisation;
- C++ adapter purposed to simplify integration with the supervision framework;
- Supervision Agents for PDR application and MAK RTI.

The description starts with a solution for supervision of the Preliminary Design Review system.

4.5.1 PRELIMINARY DESIGN REVIEW SYSTEM SUPERVISION SOLUTION

The PDR system was described in section 2.2.2. The prototype of the supervision solution for this system is purposed to evaluate applicability of the proposed concept (Web Services for system supervision), as well as interoperability aspects.

Analysing requirements for supervision framework, it was identified that the solution should provide means to monitor system usage and to detect operation failures.

This functionality is achieved using different agents, which were deployed in the system components: operating systems, EDB and GTI6-DSE servers and routes as depicted in Figure 56.

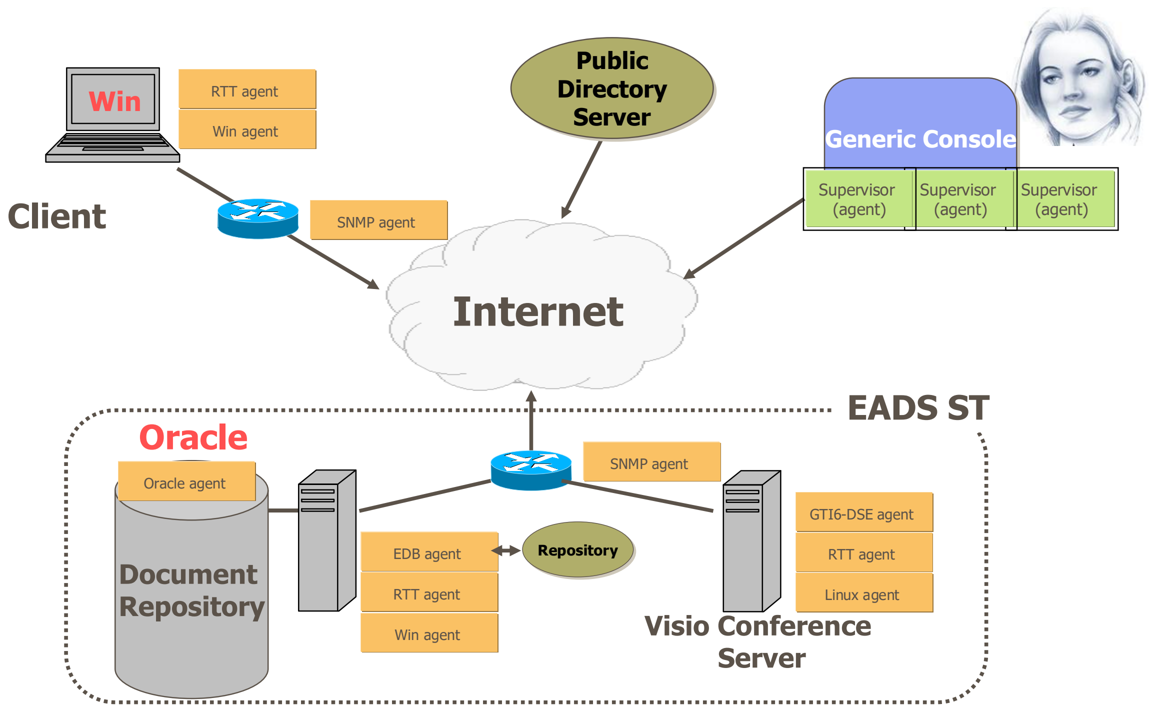


Figure 56. PDR System Supervision Solution.

The developed solution provides the monitored system parameters, which are listed in the following table. The table consists of tree columns that give information about agent capabilities, platform they are running on and means they are using to connect the supervision framework.

Agent Name	**Platform / Connection Means**	**Capabilities**
System Monitoring Agent	Windows /.NET, Linux / C++ adapter using gSOAP	Operating System Level parameters: Disk: Used Space, Free Space Memory: Free Memory, Load Ratio. Processor: CPU load (User, System), Idle time.
Network Agents	Java platforms / Java adapter using Apache AXIS	Network Level parameters: Connectivity: Round Trip Time SNMP Info
Tomcat Agent	Java platforms / Java adapter using Apache AXIS	Tomcat server info: contexts, threads, used memory.
EDB Agent	Java platforms / Apache AXIS for Java	Engineering Database (EDB) usage info: users on-line, documents created, modified, downloaded/uploaded, operation events;
GTI6-DSE Agent	Java platforms / Java adapter using Apache AXIS	GTI6-DSE video conference server info: users on-line, bandwidths allocation for each user, etc.

The following failures can be detected by the agents:

- Operating System Level failures, e.g. Disk, Memory, CPU overloading;
- Network Level failures: connectivity failures, bandwidth overloading;
- Application Level failures: document repository access violation.

In this document, the EDB monitoring solution is the author's contribution to the development of this PDR prototype.

For more information about validation process, please refer to the next chapter.

4.5.1.1 Engineering Database Agent

In the frame of the PDR scenario, the EDB Monitoring Agent collects log data from EDB Server and Clients. The agent forwards the processed data in the form of supervision messages, which are compliant with the proposed architecture.

The agent is dedicated to workflow supervision and can detect the following events:

- EDB server is started;
- User tries to enter the EDB using an EDB client;
- User enters EDB or enters a wrong password;
- User creates/reads/modifies/deletes a document;
- User leaves the EDB;
- The EDB server is stopped.

Implementation Details:

Figure 57 depicts internal structure of the Engineering Database (EDB) monitoring solution. As it was mentioned before, the EDB system consists of:

- Oracle database, which contains a Documentation Repository;
- Tomcat web-server that hosts EDB server java application;
- EDB client – a graphic user interface implementing Preliminary Design Review logic.

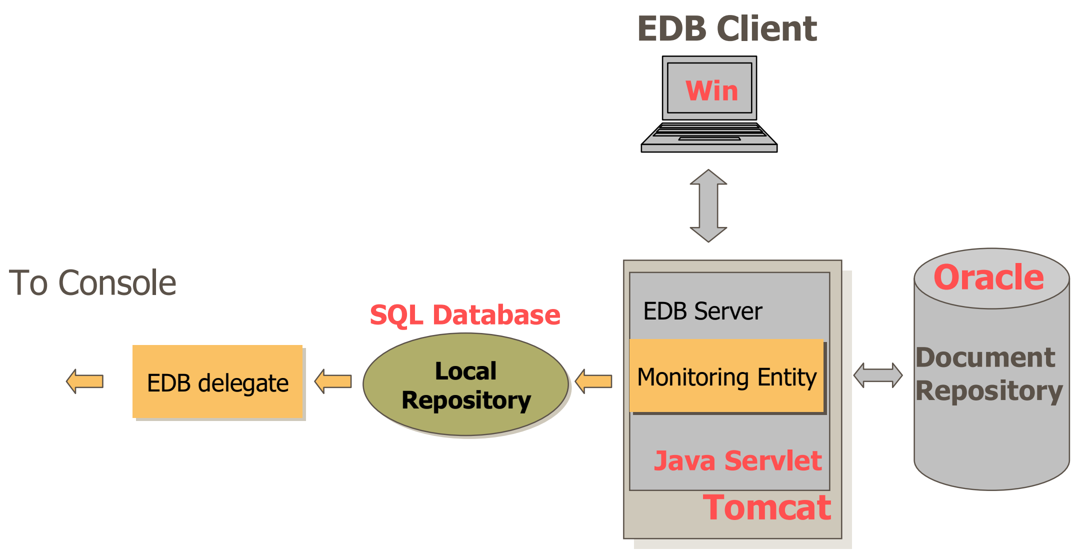


Figure 57. EDB System Monitoring Solution.

The EDB agent works directly with the EDB server java application. A monitoring entity was built-in to the server code. This module intercepts predefined server events (start-up, user login, etc.) and stores them in a Local Repository, which is a relational database. The EDB Delegate restores monitoring information from the Local Repository and forwards it to Supervisors in compliance with the supervision framework rules.

The agent handles the following monitoring information:

- **Transaction information:** An information about events happened in the Document Repository.
  - **Transaction ID:** a transaction identificator;
  - **Transaction timestamp;**
  - **Transaction owner:** an identificator of a user, who committed the transaction;
  - **Transaction mode:** description of the event (EDB server started/closed, user entered EDB, user entered a wrong password, user added a Review Item Discrepancy, user deleted a RID, user modified a RID, user downloaded a document)
  - **Transaction RID:** when the transaction concerns a Review Item Discrepancy, this parameter contains identificator of this RID.
- **User information:** recent information about users.
  - **User ID:** a user identificator;
  - **User Name;**
  - **Is the user on-line;**
  - **Last time this user entered the EDB system;**
  - **Number of times the user entered his password wrongly (to detect access violation attempts);**
  - **Last time the user entered a wrong password (to detect access violation attempts);**

- **An IP of his EDB client.**

4.5.2 Supervision Solution for Distributed Training

The Distributed Training scenario and DIS-RVM system are described in section 2.2.3. The prototype of the supervision solution for the DIS-RVM system is purposed to evaluate applicability of the proposed concept to the Interactive Simulation domain (performance issues), as well as concept flexibility to integrate simple intelligence (summarising view).

Analysing requirements for supervision framework, it was identified that the solution should provide means to monitor system usage, to detect service failures, to control systems start-up, operation and close-down. Therefore, the supervision framework and components should provide the following functionality:

- automate RTI, NTP service and DIS-RVM simulator set-up and configuration for a training session;
- provide centralised monitoring on DIS-RVM (application), RTI (middleware), OS (Linux) and network levels;
- provide Application Administrator (see section 2.4.4) with means for controlling execution of DIS-RVM and RTI;
- monitor Federation Manager, ISS, ATV, MCC, and ATV-CC federates of DIS-RVM federation.

This functionality is achieved using different agents, which are deployed in the system components: operating systems, RTI execution and DIS-RVM system as depicted in Figure 58.

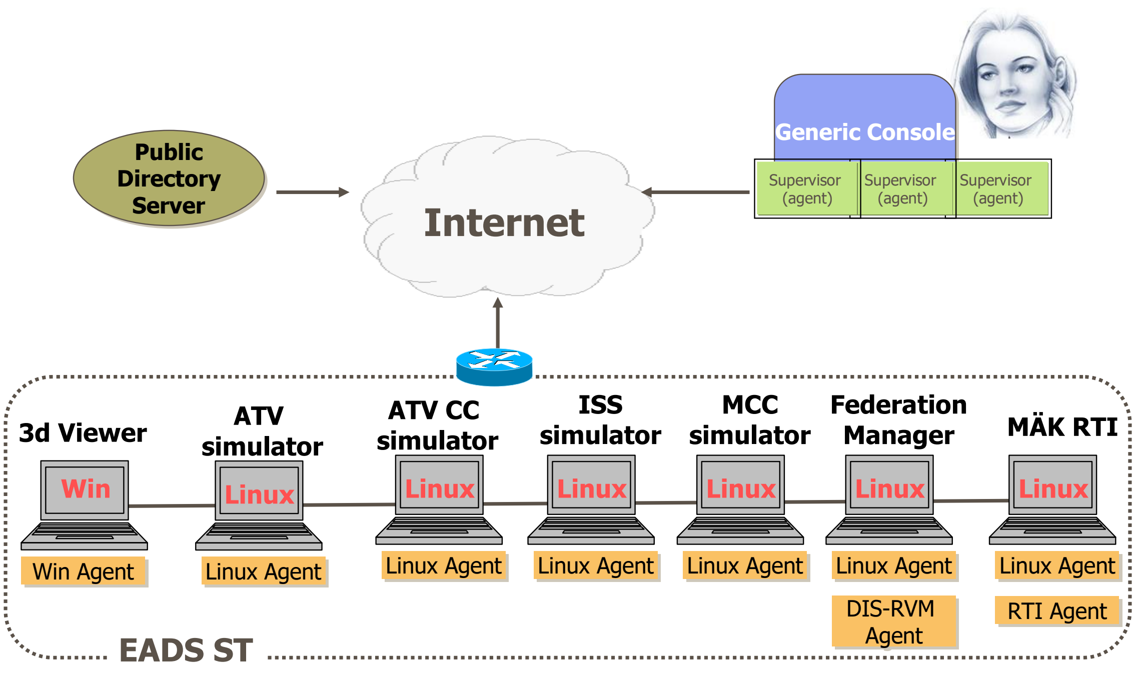


Figure 58. DIS-RVM System Supervision Solution.

The agent capabilities are listed in the following table. The table consists of tree columns that give information about agent capabilities, platform they are running on and means they are using to connect the supervision framework.

Agent Name	Platform / Connection Means	Capabilities
System Monitoring Agent	Windows /.NET, Linux / C++ adapter using gSOAP	Operating System Level parameters: Disk: Used Space, Free Space Memory: Free Memory, Load Ratio. Processor: CPU load (User, System), Idle time.
System Control Agent	Linux and Windows / C++ adapter using gSOAP	Control Agent is enabled to configure RTI, NTP, DIS-RVM and to control applications execution running predefined system commands.
MÄK RTI Monitoring Agent	Linux / C++ adapter using gSOAP	To monitor RTI Execution state, Federation and Federate states.
DIS-RVM Monitoring Agent	Linux / C++ adapter using gSOAP	To monitor NTP synchronisation, Federate states and to control simulator execution.

Thus the following failures are detected in addition to system and network failures:

- Abnormal federate termination;
- NTP is out of synchronisation.

To effectively manage the DIS-RVM system, a generic console with a summarising interface was built to present system health status and all critical parameters in a single view. The console capabilities are as follows:

- visualisation of all agent parameters described in a special format (see next section);
- filtering of incoming messages using regular expression rules;
- creation of graphic plots;
- summarising GUI for DIS-RVM system (see intelligence section).

For more information about validation process, please refer to the next chapter.

The following sections give more details about the developed approaches and some components worked out by the author of this thesis as a contribution to the DIS-RVM prototype.

4.5.2.1 Agent Collaboration for Generic Visualisation

Although the operation description mechanism (section 4.4.5) implies a generic format for supervision data description sufficient for supervision framework, certain data processing operations require more precise instructions. For example, supervision data visualisation requires a comprehensive description of:

- operations (name, readable description, etc);
- parameters (name, readable description, units, measurement methods, etc);
- visualisation rules (lists, tables, graphics, charts, etc.).

The flexibility of the proposed architecture allows centralised storing of these custom descriptions with the agent information. Thus, all interested agents have an access to this information.

Developing general format for visualisation rules description allowed building of a generic console for DIS-RVM system, which can seamlessly work with the agents that provide a correct description.

Implementation Details:

All the agents involved in the Distributed Training Scenario support a generic format for description of visualisation rules as follows.

Similar to supervision messages, the visualisation rules are encoded using XML, in the following format:

`<?xml version="1.0" encoding="UTF-8"?>` `<operation>`	Starting description
`<displayName>Some Operation</displayName>`	Name of the monitoring category to be shown in the console
`<description>Provides comprehensive information about some monitoring category </description>`	Each parameter has a textual description for the end-users.
`<request>`	Starting description of the input parameters.
`<parameter>`	Supervisors declare parameters to be passed to Delegates with the request for this information. For example, for disk monitoring operations, we can precise disk ids for which the information is required (“/”,”/home”,”/usr” or ”C:”, “D:” and etc)
`<name>memAvaMain</name>` `<displayName>Available Memory</displayName>` `<description>Size of availble RAM in bytes.</description>`	Here, a parameter name is declared. **displayName** tag contains parameter name to be shown by the Console, while **name** tag indicates an internal name of the parameter used in message construction and in processing.
	description tag contains parameter description in a human readable form.
`<type>SomeType</type>` or `<list>` `<type>SomeType</type>` `</list>`	**type** tag indicates a parameter type. Analysing the monitoring data for the Distributed Training scenario and in order to simplify parsing of the description, the used types are:
	int – C style integer type
	float – C style float type
	string – any string

Optionally, list tag can indicate multiple value entries. For example for a Federation Name list:

```
<parameter>
    <name>FederationList</name>
    <list>
        <value>FirstFederation</value>
        <value>SecondFederation</value>
        <value>ThirdFederation</value>
    </list>
</parameter>
```

`</parameter>`	End of the parameter declaration block
…	Other input parameters
`</request>`	End of input parameters block.
`<response>`	Starting description of the output parameters
`<parameter>…</parameter>` …	Declarations of output parameters in the same way as for the input parameters
`</response>`	End of output parameters block

This operation description allowed to set-up generic interface for all operations (Monitoring Categories, Control operations, etc) in Distributed Training scenario. Thus the console can seamlessly create an interface for any agent, which describes its capabilities in this form.

For instance, the System Monitoring Agent describes Memory Information operation as follows:

```
 <?xml version="1.0" encoding="UTF-8"?>
<operation>
    <displayName>Memory Information</displayName>
    <description>Provides information about Memory</description>
    <request>
        <!-- there is no special input parameters -->
    </request>
    <response>
        <parameter>
            <name>memAvaMain</name>
            <displayName>Available Memory</displayName>
            <description>Size of availble RAM in bytes.</description>
            <type>float</type>
        </parameter>
        <parameter>
            <name>memLoadRatio</name>
            <displayName>RAM load ratio</displayName>
            <description>Load ration of RAM.</description>
            <type>float</type>
        </parameter>
        <parameter>
            <name>memAvaSec</name>
            <displayName>Available secondary memory</displayName>
            <description>Size of availble secondary memory in bytes.</description>
            <type>float</type>
        </parameter>
        <parameter>
            <name>memSecLoadRatio</name>
            <displayName>Secondary memory load ratio</displayName>
            <description>Load ration of secondary memory.</description>
            <type>float</type>
        </parameter>
        <parameter>
```

```
            <name>HelthStatus</name>
            <displayName>Health Status</displayName>
            <description>Health Status parameter indicates status of the monitoring agent for monitoring information
evaluation and summarising, according the summariser monitoring approach. Its value is a choice between
green|yellow|red. </description>
            <type>string</type>
        </parameter>
    </response>
    </operation>
```

4.5.2.2 Intelligence – Summarising View

The supervision of complex industrial systems often requires intelligent solutions. The intelligence can be provided in several ways, some of which are outlined here:

- **Hard coded behaviour**: an "intelligent" reaction on a system status can be implemented as a part of the program code of an agent.
- **Parameter based 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 the events received.
- **Trends prediction:** A behaviour pattern recognition system could be developed to react in advance according to system behaviour trends.

Besides, working with hundreds of critical parameters is an extremely difficult task. An administrator can be easily flooded with low level warnings like "memory is running low" or "maximum number of users is almost reached". Instead, the administrator first needs a general, summarised view about the health of the systems in general and then can go into details as necessary.

The proposed architecture is flexible to combine supervision messages with the "health" state evaluation metrics. This feature is used in the Distributed Training scenario to illustrate flexibility of the proposed supervision concept in the context of intelligence.

Implementation Details:

In the Distributed Training Scenario a simple intelligence solution was developed. This solution implements the summarising approach. The DIS-RVM system involves several hosts, where many critical parameters should be evaluated to summarise the system health status.

The monitoring information is evaluated both on Delegate and Supervisor levels. The agents provide specific status evaluation parameters that can be symbolically divided into 3 categories: green – best state (=1), yellow – satisfactory state (=2), red – failure state (=0). The delegates calculate their status according to some predefined rules. For example system=>memory state could be defined by thresholds (green, when memory load ratio is less than 80%, yellow, when ratio is greater than 80%, but less than 90%, red, when the ratio is greater than 90%). The Summary state is calculated in the Console, as a simple convolution of all the states.

In this way, the summary state can be expressed by the following equation:

$$Global\_Status = \prod_{all\_monitoring\_categories} monitoring\_category\_status = \prod_{all\_agents} agent\_status = \prod_{all\_agents\_and\_all\_parameters} parameter\_status = \begin{cases} 1 \Rightarrow green \\ >1 \Rightarrow yellow \\ 0 \Rightarrow red \end{cases}$$

The Figure 59 illustrates this equation.

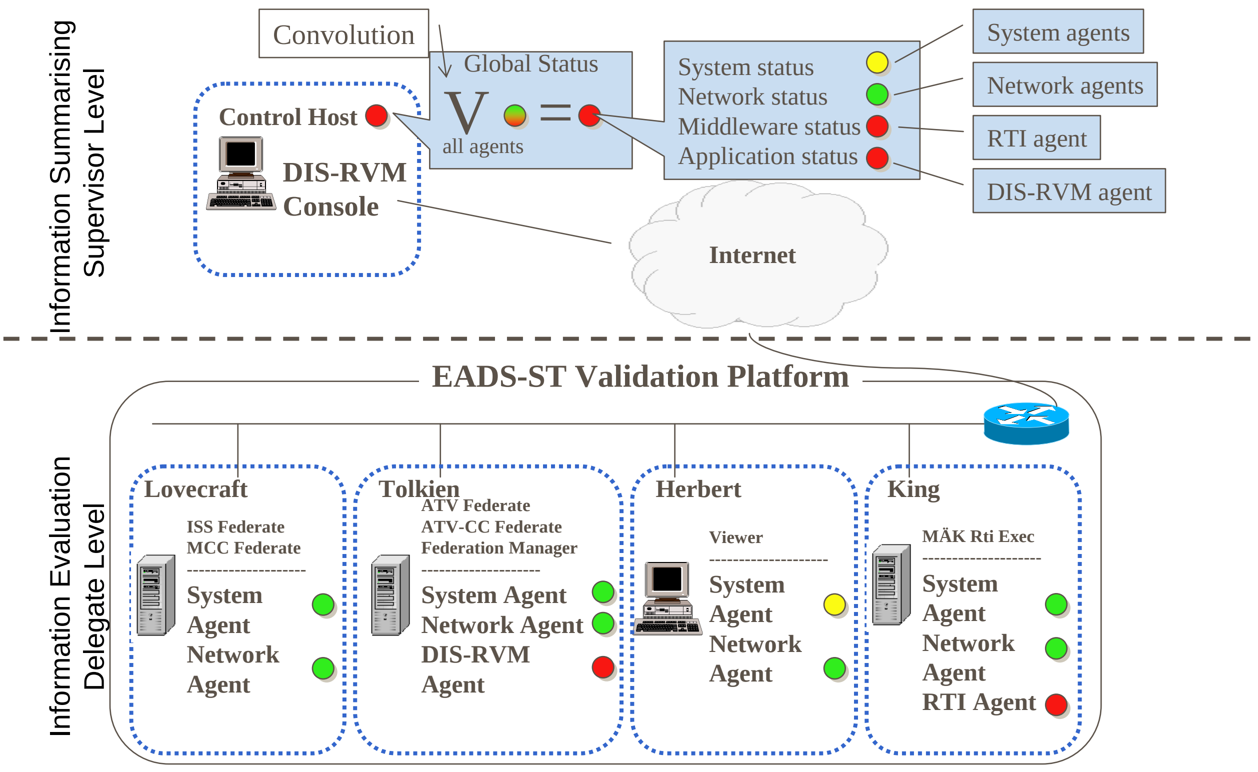


Figure 59. Information Evaluation-Summarizing in Distributed Training Scenario.

The information is summarised in the Console, the Global Status is composed of Monitoring Category Status (System, Network, Middleware, and Application) and it, in its turn, is composed of agent status (System, Network, RTI, and DIS-RVM). Consequently, the yellow system agent status at Herbert causes the System Status to be yellow, while red status of the DIS-RVM and RTI agents cause Middleware, Application Statuses and, thus, Global Status to be red.

The agents provide this information in the form of message parameters. For the Memory Health Status the operation declaration looks like as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<operation>
    <displayName>Memory Information</displayName>
    <description>Provides information about Memory</description>
    <request>
        <!-- there is no special input parameters -->
    </request>
    <response>
        <parameter>
            <name>memAvaMain</name>
            <displayName>Available Memory</displayName>
            <description>Size of availble RAM in bytes.</description>
            <type>float</type>
        </parameter>
        <parameter>
```

```
            <name>memLoadRatio</name>
            <displayName>RAM load ratio</displayName>
            <description>Load ration of RAM.</description>
            <type>float</type>
        </parameter>
        <parameter>
            <name>memAvaSec</name>
            <displayName>Available secondary memory</displayName>
            <description>Size of availble secondary memory in bytes.</description>
            <type>float</type>
        </parameter>
        <parameter>
            <name>memSecLoadRatio</name>
            <displayName>Secondary memory load ratio</displayName>
            <description>Load ration of secondary memory.</description>
            <type>float</type>
        </parameter>
        <parameter>
            <name>Health Status</name>
            <displayName>Health Status</displayName>
            <description>Health Status parameter indicates status of the monitoring agent for monitoring information
evaluation and summarizing, according the summarizer monitoring approach. Its value is a choice between
green|yellow|red. </description>
            <type>string</type>
        </parameter>
    </response>
    </operation>
```

Therefore, each Memory Information message contains the Memory Health Status, as illustrated in the following example:

```
    <response>
        <parameter>
            <name>memAvaMain</name>
            <value>325454387.0</value>
        </parameter>
        <parameter>
            <name>memLoadRatio</name>
            <value>98.0</value>
        </parameter>
        <parameter>
            <name>memAvaSec</name>
            <value>0.0</value>
        </parameter>
        <parameter>
            <name>memSecLoadRatio</name>
            <value>0.0</value>
        </parameter>
        <parameter>
            <name>HealthStatus</name>
            <value>red</value>
        </parameter>
    </response>
```

In this way, all agent messages contain evaluation information. Then, the DIS-RVM console summarises the whole status and shows it to an operator (Figure 60).

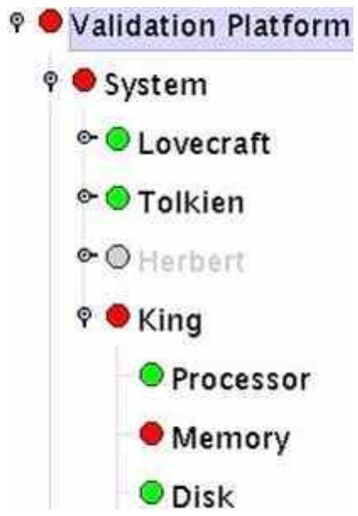


Figure 60. A DIS-RVM Console Screenshot.

4.5.2.3 C++ ADAPTER

C++ Adapter was developed to simplify agent creation. It takes charge of service operations and allows developers to concentrate on supervision logic. In these terms, the adapter represents the major component of the supervision middleware.

The adapter functionality involves all the aspects of an agent life-cycle in the supervision framework and includes:

- Automatic Registration / Update Registration / Unregistration in the supervision framework;
- Management of Query requests;
- Management of Subscribe requests, including handling of the list of subscribers;
- Handling of periodic events;
- Notification Service for handling of asynchronous events;
- XML Factory Service to create and parse supervision messages;
- SOAP server based on gSOAP library engine.

The following scenario agents use this C++ Adapter:

- Linux operating system agent and Windows control agent;
- DIS-RVM agent;
- MÄK RTI agent.

From the developer's point of view, the adapter represents a stand-alone component and an API to connect with. The implementation details are provided hereafter.

Implementation Details:

The following UML class diagram (Figure 61) shows the overall structure of the C++ Adapter.

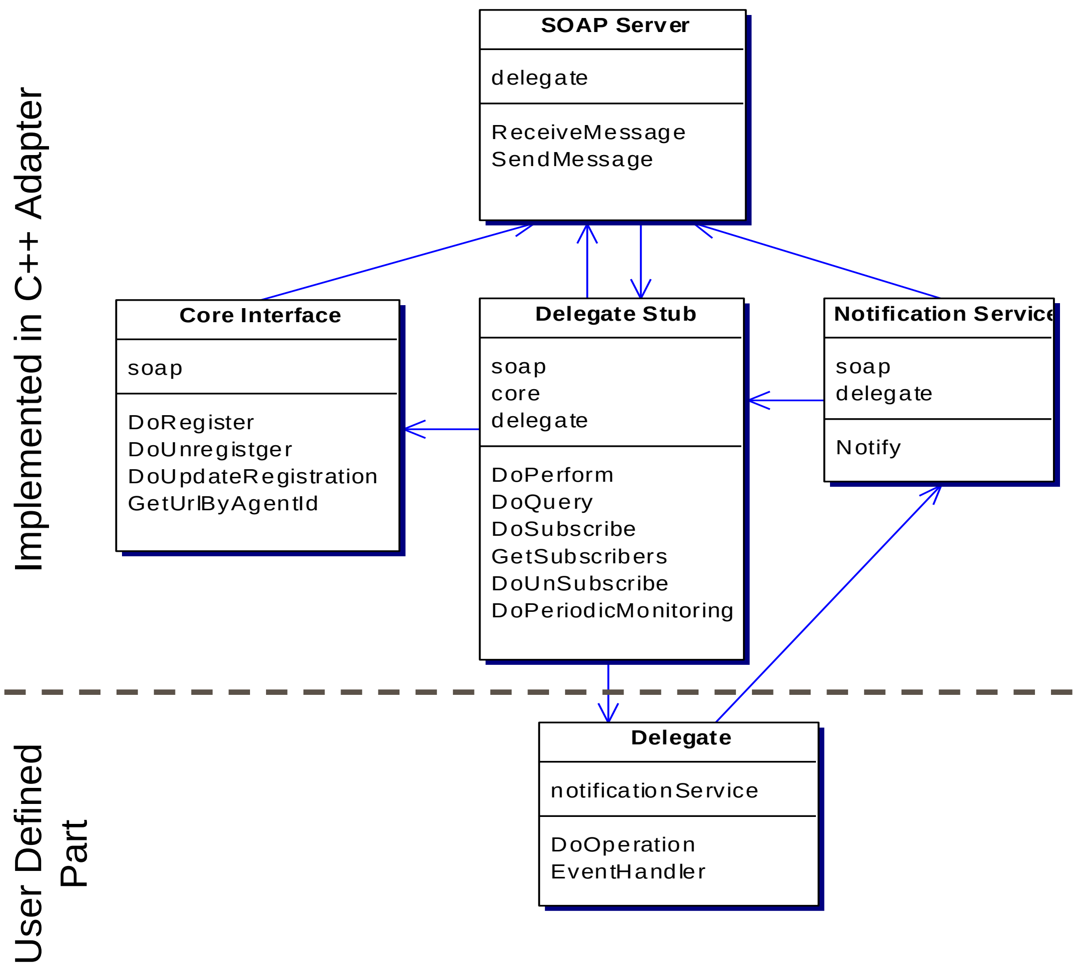


Figure 61. C++ Adapter – UML Class Diagram.

The adapter has the core interface for service operations and implements the delegate paradigm for supervision operations. The Notification Service class provides a mechanism of an asynchronous communication with a supervisor.

Agent developers implement supervision logic in the stand-alone Delegate module. The predefined methods (**DoOperation, EventHandler**) are the place where developers should put their code to process supervision operations. **EventHandler** method uses the Notification Service to transmit asynchronous events to a supervisor.

During start-up the adapter registers the delegate in the Core using Core Interface, which in its turn uses SOAP server engine for communication with the Core.

Upon a supervision request the SOAP server process an incoming message. Depending on the type of the message the server passes this request to a corresponding method of the Delegate Stub.

Upon query/perform request, the DelegateStub invokes Delegate's **DoOperation** method, which returns output information, which then is encoded in the DelegateStub and is passed back to a supervisor.

Upon a subscribe/unsubscribe request, the DelegateStub handles a subscriber list. If there is a request on periodic supervision, **DoPeriodicMonitoring** method periodically invokes **DoOperation** method.

Upon an event, the Delegate invokes the **Notify** method of the Notification Service, which, in its turn, verifies subscribers for this event, encodes the message and sends it via SOAP server.

For a better performance, the adapter generates messages using simple patterns. This permits to avoid DOM model that highly consumes computation resources.

4.5.2.4 MÄK RTI AGENT

The MÄK RTI is an implementation of the HLA RtiExec service [7] purposed to run interactive simulations. This software was used in the Distributed Training scenario to run the Rendezvous Federation mentioned in the introduction concerning industrial problems.

As it was mentioned above, the Distributed Training scenario requires a permanent supervision of all the levels (system, network, middleware, application). The RTI Agent (Delegate) was designed to provide supervision functionality on the middleware level (RtiExec). Besides, the RTI Delegate is a general purpose agent that can be used with any federation that does not utilise Lightweight Mode of MÄK RTI [53].

The agent functionality includes:

- **RtiExec state monitoring.** The RTI Delegate detects whether RtiExec is running or not. It also processes an event on RtiExec load/close down.
- **Federation monitoring.** The delegate retrieves a list of registered federations and notifies about federation creation/destruction.
- **Federate monitoring.** The agent retrieves a list of the registered federates for each available federation and notifies on federate joining/resigning.
- **Connection Errors.** The agent notifies when a connection between an RtiExec and Federate is abnormally closed.

The following section describes implementation details.

Implementation Details:

The RTI Supervision Solution consists of two independent modules, as depicted in Figure 62 :

- **RTI Delegate** communicates with the supervision framework via C++ Adapter for the Linux platform. It is an independent module that is implemented outside of MÄK RTI code with the aim to monitor Rti Execution state;
- **RTI Monitoring Plugin** is built in the MÄK RTI and monitors Rti execution.

These two modules communicate using a custom protocol, which is purposed to transmit Rti heartbeats (for a state detection), supervision commands (a query for particular monitoring data) and asynchronous events (state, federation and federate events).

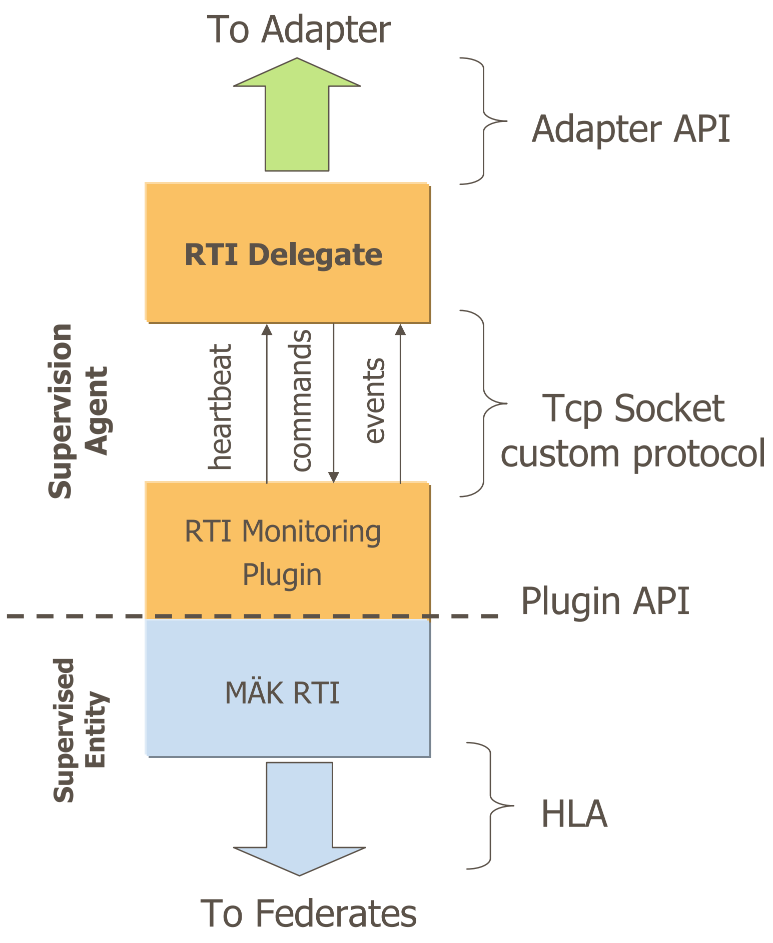


Figure 62. MÄK RTI Delegate Overall Architecture.

The RTI Monitoring Plugin uses MÄK RTI Plugin API [53] that allows attaching additional handlers to HLA events, as well as retrieving of particular information concerning Rti Execution.

Meanwhile, the RTI Delegate collects the monitoring information and communicates with the supervision framework (DIS-RVM Console) via the C++ adapter described in the previous section.

The RTI Delegate provides the following supervision operations:

- Query operations:
  - Rti State: State of the Rti Execution indicates whether Rti is ON or OFF;
  - Federation List: Provides a list of available federations;
  - Federate List: Provides a list of available federates.
- Event-based subscribe operations:
  - Rti State Event: happens when Rti Execution gets ON or OFF;
  - Federation Event: happens when a federation is created or destroyed in the Rti Execution;
  - Federate Event: happens when a federate joins or resigns a federation

- Connection Dropped Event: happens when a Federate abnormally closes a link to a Federation.

These operations provide all the necessary information to monitor MÄK Rti Execution in the Distributed Training scenario.

4.5.3 CONCLUSIONS

In this section a prototype solution has been presented. Two supervision solutions were developed to evaluate the proposed concept in the context of the industrial problems. These solutions implement all major elements of the proposed architecture (The Core – Directory Server, Delegate, and Supervisor) and several approaches for intelligent supervision.

The C++ adapter developed by the author provides a simple way to create agents, while EDB and MÄK RTI agents are application specific and intended to help to evaluate the solutions.

The next chapter describes the evaluation process.

Chapter 5. EVALUATION

The goal of evaluation activities is to verify applicability of the proposed solution to the domain of distributed supervision. In this way, the prototype developed for completely different distributed applications should show flexibility and comprehensiveness of the proposed concept. The evaluation section starts with the development lessons learned to evaluate the concept from a developer's point of view. The development permitted to evaluate the following requirements:

- Interoperability and Portability;
- Flexibility;
- Ease of Development.

Then, the performance tests section gives an idea about overall performance of the supervision middleware in the case of Distributed Training supervision. In addition, these activities permitted to estimate the following issues:

- Impact of the supervision agents to operating system and network load;
- Applicability to supervision of the DIS-RVM system;
- Limits of applicability;
- Fault Tolerance and Reliability.

Finally, the validation scenario sections elaborate the concept evaluation from an end-user's point of view. In the industrial domain, it is widely accepted to hold user validation, when real users participate in real-life validation sessions. During these sessions, users imitate normal work playing scenarios, and evaluate an impact of new enhancements on the system.

This process clearly shows the features that are difficult to "calculate":

- Applicability;
- Impact of the Supervision Solution to the Overall System Performance;
- Comprehensiveness ;
- Fault Tolerance concerning platform faults inserted by users;
- Ease of Deployment;
- Usability.

The subsequent section elaborates development issues during prototyping.

5.1 DEVELOPMENT LESSONS LEARNED

While developing the prototype, at first, Web Services interoperability issues arose. A common practice in the Web Services domain is to design a WSDL description for each service and then to use special toolkits to create a client and server code. The tests showed that for the simple services this mechanism generates interoperable code for different kinds of programming languages: C/C++, C#, Java, PHP [51]. However, in complex cases, developers use low-level SOAP features, like literal encoding. As a result, the interoperability of automatically generated code is questionable. While implementing the current architecture, the .Net, Axis and gSOAP toolkits showed lack of interoperability in regard to encoding of custom message headers and a literal message body. This problem was solved during development of the adapters, which provide an interface for the supervision framework. This permitted to avoid these interoperability issues in a further development. In addition, development of the adapters permitted to diminish efforts for an agent development.

From a developer’s point of view, usability is assured by the adapters and sample agents, which provide means to rapidly develop new agents. At the same time, these means simplify concept learning for newcomers. For example, the C++ adapter provides an empty agent that implements all service operations and is ready to integrate with a supervised entity. Therefore, development duration depends on complexity of interface implementation, integration and testing. For the agents of DIS-RVM supervision system, it took about 1-2 weeks to integrate the supervised entities (operating system, MÄK RTI, DIS-RVM) with the C++ adapter. Taking into account that this integration was performed during experimentation with C++ adapter and its debugging, it should be pointed out that under normal conditions implementation of the interface is supposed to take even less time.

The proposed architecture proved its flexibility, since all the required types of data and supervision operations were successfully implemented into the prototype. For example, the MÄK RTI monitoring agent provides information about federations and federates in a form of a list. Besides, this agent implements an event based subscription, which proves an ability to implement asynchronous communication by means of Web Services. In comparison with the periodic subscription implemented in the System Control, Monitoring and DIS-RVM agents, the event-based subscription (MÄK RTI Agent) can significantly diminish the network load, which is the most critical point of Web Services. Additionally, development of approaches for generic visualisation and for information summarising resulted into a ready-to-use solution for an intelligent supervision. This is because the developed Supervision Console supports any custom agents that implement the above-mentioned approaches. Therefore, all these aspects contributed to simplification of the development of new agents.

The developed prototype revealed portability of the architecture, allowing interoperation of the components programmed in C++, C#, Java, PHP, which had been intended to be used in different operating systems, such as Windows and Linux. For instance, the System Control agent designed for Linux and developed in C++ was successfully ported to the Windows platform. Also, a PHP based agent which can be run on any Web server that supports PHP scripts, while Java-based agents can be run on any Java compliant platform. This allows developers to choose a programming platform according to their needs.

It should be mentioned that the Web Services performance has always been an inevitable question of this technology, since XML parsing is a heavy task for computation resources. As it was mention before, there are two most widely used methods of XML parsing, DOM and SAX. To alleviate performance issues the C++ adapter uses a parser similar to SAX, which has incontestably better characteristics in regard to performance. The following section elaborates these aspects concerning the supervision solution for the DIS-RVM system. Besides, the limits of the concept applicability will also be considered there.

5.2 PERFORMANCE TESTS

The performance tests evaluate the major parameters that can affect applicability of the supervision middleware:

- System resource load and, in particular, CPU load;
- Network load and Throughput;
- Response time.

These parameters were measured for the DIS-RVM system supervision solution, since the distributed simulation, in this case, has strict performance requirements:

- Latency of 300 milliseconds
- Bandwidth of 64 kbps

Therefore, the main goal of the performance tests is to evaluate a possible impact of the supervision on calculation environment.

The following section describes measurement means and approach.

5.2.1 MEASUREMENT APPROACH

The performance tests bring out a question of how using of the Web Services for supervision affects the supervised environment. The below-described measurements are intended to answer this question.

The test platform consists of three hosts: Test Tool Host, Agent Host, Network Spy depicted in Figure 63.

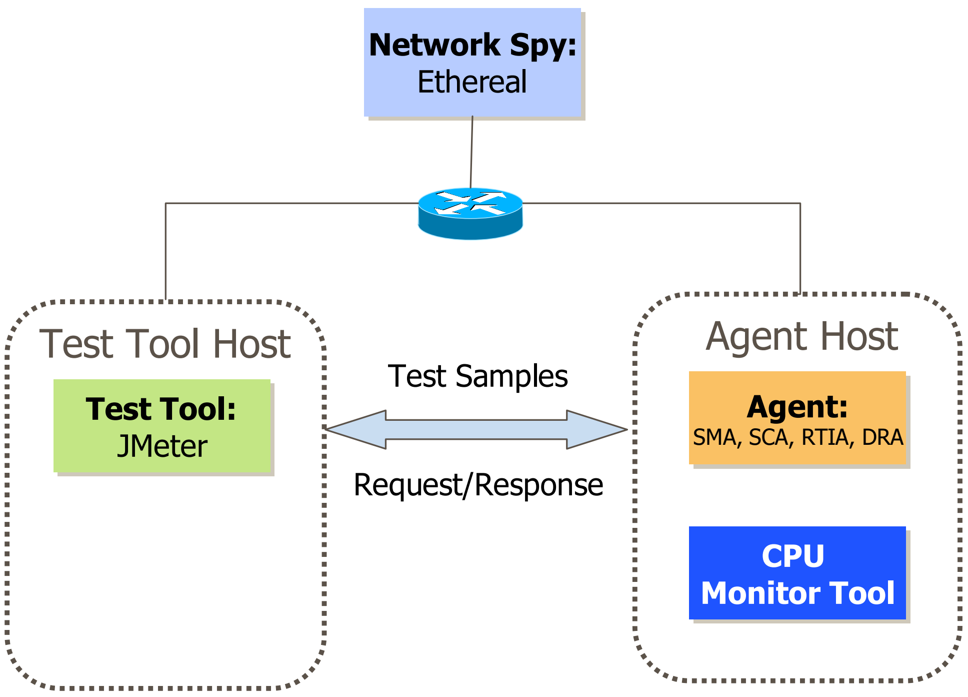


Figure 63. Performance Test Platform

As a test tool, Apache JMeter [54] provides means to send message samples and to measure the response time. It returns results in the form of an XML file, which are analysed lately. For the test purposes, the JMeter issued only heartbeat request samples. This permitted to better evaluate supervision middleware performance characteristics, since this operation does not involve interaction with a supervised entity. Thus, these tests show efforts to marshal and to unmarshal XML messages only. In other words, the agent responsiveness, in this case, is affected by the XML parsing only and, therefore, shows efficiency of the middleware components (C++ adapter).

The following heartbeat message request illustrates test samples:

```
POST / HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: Axis/1.1
Host: 212.234.91.65
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Content-Length: 1756

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Header>
  <ns1:genesysHeaderBlock soapenv:mustUnderstand="1" xsi:type="ns1:GenesysHeaderBlock" xmlns:ns1="http://genesys-
project.org/GMP">
   <ns1:sourceComponentId xmlns="http://genesys-project.org/GMP">
    <ns1:coreId>1</ns1:coreId>
    <ns1:ownerId>0</ns1:ownerId>
    <ns1:siteId>0</ns1:siteId>
    <ns1:agentId>e473fbcd-ffffff81-15d533d-2b52c35e</ns1:agentId>
   </ns1:sourceComponentId>
   <ns1:destinationComponentId xmlns="http://genesys-project.org/GMP">
    <ns1:coreId>1</ns1:coreId>
    <ns1:ownerId>e44b6f1e-ffffff81-11a0070-947524fd</ns1:ownerId>
    <ns1:siteId>e44cf3da-ffffff81-11a0070-45735803</ns1:siteId>
   <ns1:agentId>e4744809-ffffff81-15d533d-588d3b2d</ns1:agentId>
   </ns1:destinationComponentId>
   <ns1:timestamp xsi:type="xsd:long">1099049882225</ns1:timestamp>
  </ns1:genesysHeaderBlock>
 </soapenv:Header>
 <soapenv:Body>
  <ns2:Query soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:ns2="http://genesys-
project.org/GMP">
   <monitoringEncodingName xsi:type="xsd:string">http://genesys-
project.org/xsd/encoding/gse/GenesysEncoding</monitoringEncodingName>
   <monitoringRequest xsi:type="xsd:string">&lt;?xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot;?&gt;&#xd;
&lt;supervisionRequest xmlns=&quot;http://genesys-
project.org/xsd/encoding/gse/GenesysEncoding&quot;&gt;&lt;typeName&gt;http://genesys-
project.org/xsd/monitoring/gse/common/Heartbeat&lt;/typeName&gt;&lt;/supervisionRequest&gt;&#xd;
</monitoringRequest>
  </ns2:Query>
 </soapenv:Body>
</soapenv:Envelope>
```

To summarise, the JMeter tests provided information about response time and throughput (number of complete request/response interactions per second). These parameters were measured under the following conditions:

- **Normal load:** the JMeter requests an agent once per 20 milliseconds (1000 cycles in total for each agent);
- **Heavy load:** 20 concurrent test components request an agent once per 20 milliseconds (20000 cycles in total for each agent).

It should be mentioned, that that the provided frequencies are JMeter settings only. The actual frequencies are characterised by the throughput parameter. In reality, the JMeter uses a best-effort approach, i.e. it sends samples as fast as it can. Hence, the throughput parameter is an important metric to evaluate real performance.

All tests were conducted separately for each agent involved in the DIS-RVM scenario (System Monitoring, System Control, MAK RTI and DIS-RVM agents).

The Agent Host has the following characteristics:

Operating system:	Mandrake Linux 10.0
Kernel:	2.4.26-mdk
CPU:	Inter Pentium IV 1,80 GHz
Network connection:	LAN 10Mbit
Memory:	384 MB of RAM

In order to measure how the supervision system affects the local operating system, the native Linux tools measured the CPU load on the Agent host.

Meanwhile, the Ethereal Network Protocol Analyser [55] registered the incoming/outcoming network traffic for the Agent Host. This data allowed calculating of the real bandwidth statistics.

The following section presents the measurement results for the agents involved in Distributed Training scenario.

5.2.2 TEST RESULTS

As it was mentioned before, the major parameters that were gathered are:

- Response time (for the Heartbeat Operation);
- Throughput (number of complete request/response interactions per second);
- Network load (in kbits per second);
- CPU load on the Agent Host.

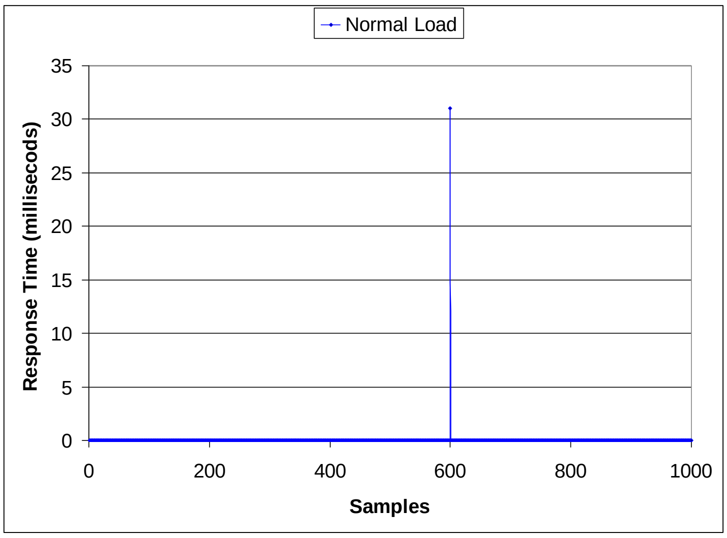
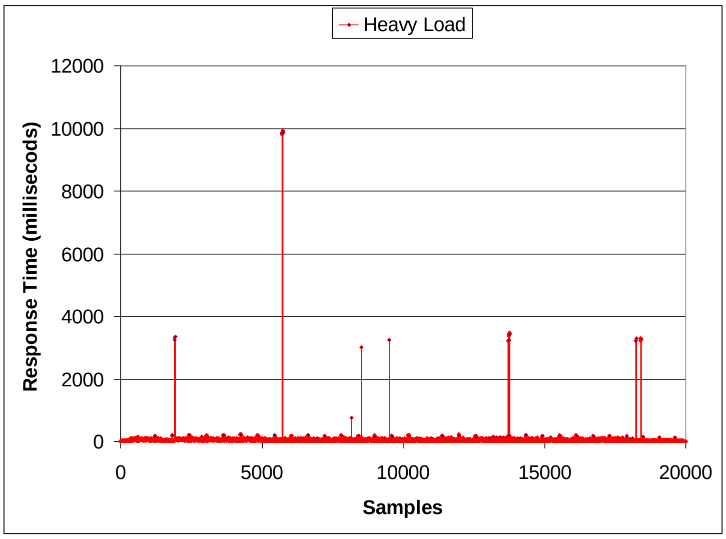
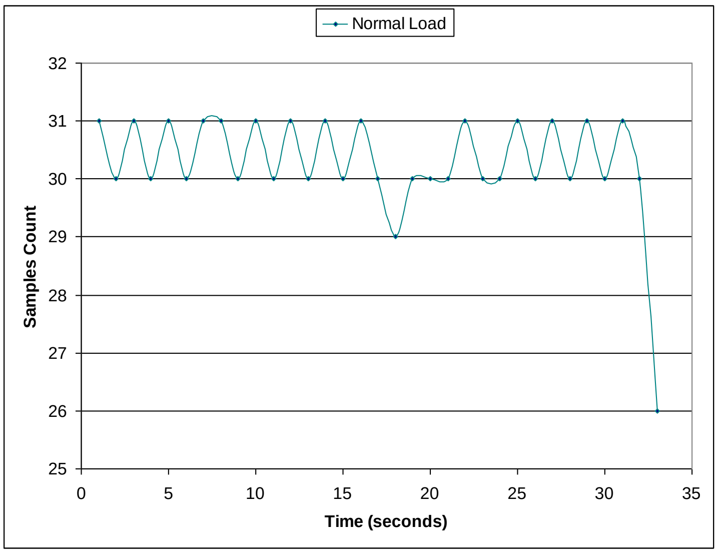
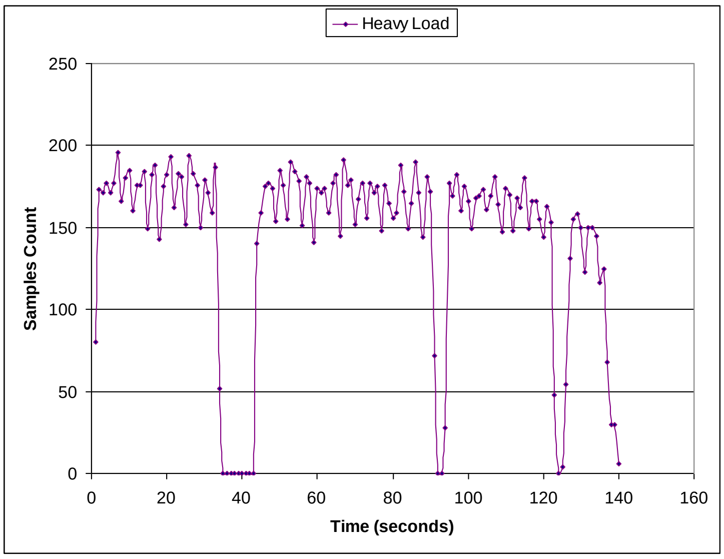
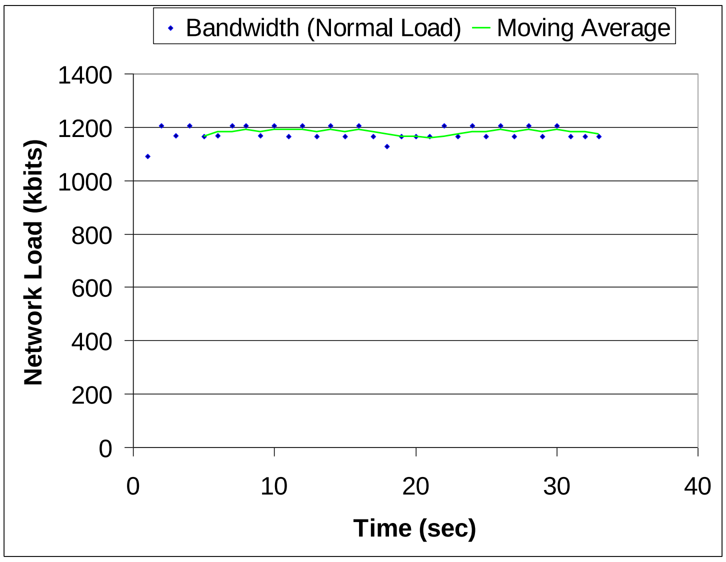
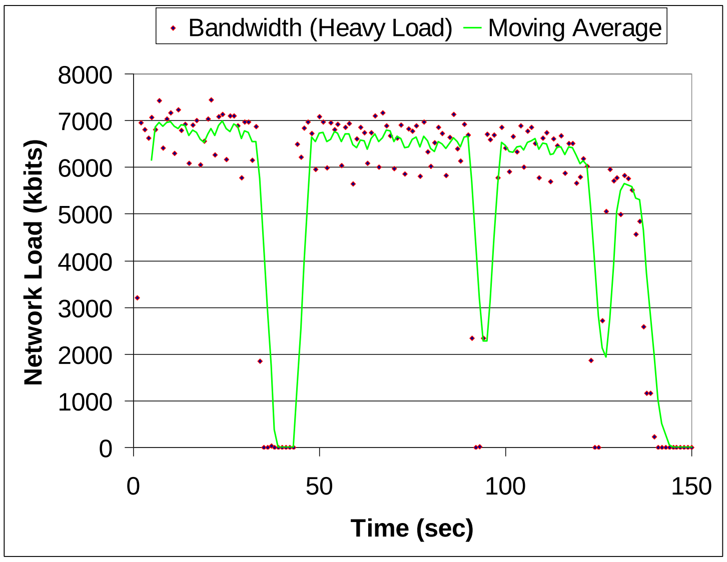
The following subsections show this statistics for every agent involved in Distributed Training scenario.

5.2.2.1 SYSTEM MONITORING AGENT

The following table points out the average and maximum values in sets of measured parameters:

Category	Average	Maximum
Response time (in milliseconds):		
Normal Load (1000 cycles)	0,031	31
Heavy Load (20000 cycles)	59,13	9922
Throughput (samples/second):		
Normal Load (1000 cycles)	30,27	31
Heavy Load (20000 cycles)	142,85	196
Network Load (kbits/second):		
Normal Load (1000 cycles)	1073,37	1205,37
Heavy Load (20000 cycles)	5116,57	7436,3
CPU load (in %):		
Normal Load (1000 cycles)	5,81	7,9
Heavy Load (20000 cycles)	32,52	38,5

The following plots depict performance measurements for the System Monitoring Agent.

Normal Load	Heavy Load
	
	
	

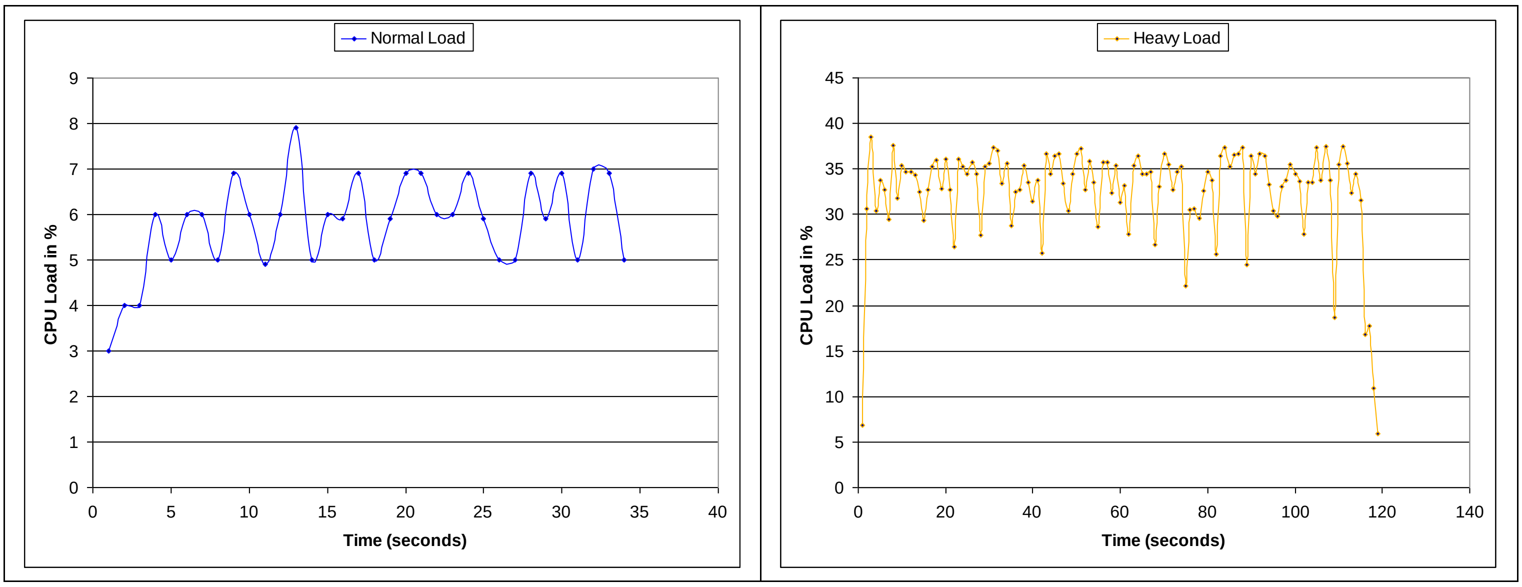


Figure 64. System Monitoring Agent – Performance Measurements

These results show good performance characteristics of the System Monitoring Agent. Even for the heavy load case, the average response time was no more than 60 milliseconds. However, there were several peaks taking more than three (3) seconds for a response. In the meantime, in the normal case the response time was always acceptable in regard to the latency requirement for the DIS-RVM system. Besides, the response time was mostly less than one millisecond, which is comparable to responsiveness of middleware using binary protocols.

Concerning the throughput, the normal use-case pointed out to an ability of the agent to successfully manage at least 30 messages per second. This result seems to be quite acceptable, since it is planned to use maximum two consoles in the real supervision system.

It should be highlighted that, as tests showed, JMeter was capable to process up to 200 outcoming/incoming message pairs only. Thus, the throughput parameter shows actual number of interactions. The measurements showed that under heavy load conditions, JMeter could stop for some moment to process all pending messages. This problem surely concerns implementation of JMeter, since it is a multi-thread Java application. However, it noticeable, that the agent itself is quite faster than the tester, considering that there was no lost messages signalled by the JMeter.

According to network load measurements, request/response sample takes about 35,82 kbits in total. This parameter was calculated as a ratio of the average network load parameter to the average throughput. Considering that a timestamp itself takes 32 bits, the overhead looks enormous. It is a predictable result in the XML world, however the tests show real evaluations of the consumption. In the LAN, this overhead is rather acceptable, but it is obviously not the case for the DIS-RVM system, which requires a bandwidth of 64 kbits/s. Thus, the supervision solution is good enough for using in LAN or broadband WAN. Tests of network load for the whole supervision system will be provided in the section 5.4.2, which describes validation sessions of the Distributed Training Scenario. These measurements will indicate real bandwidth consumption.

As for the CPU load tests, the agent showed suitable results for the both normal and heavy load cases. The CPU load did not exceed 8% with the normal load and 40% with the heavy load. However, one can consider moving heavily loaded components like a supervision console to a separate PC.

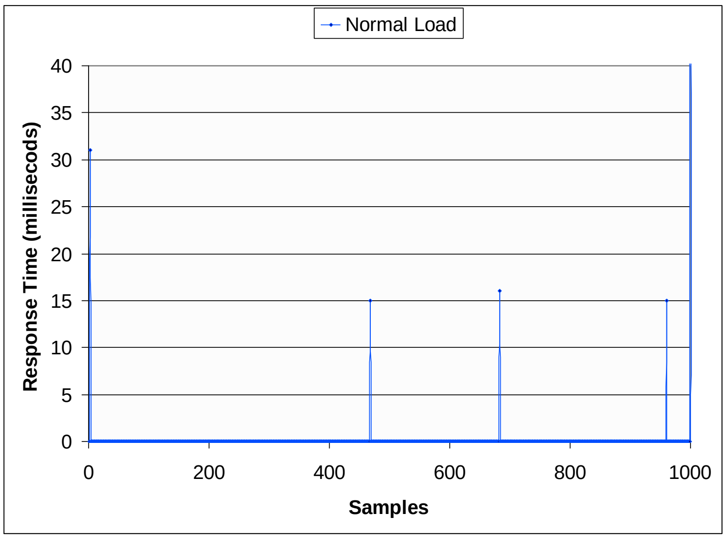
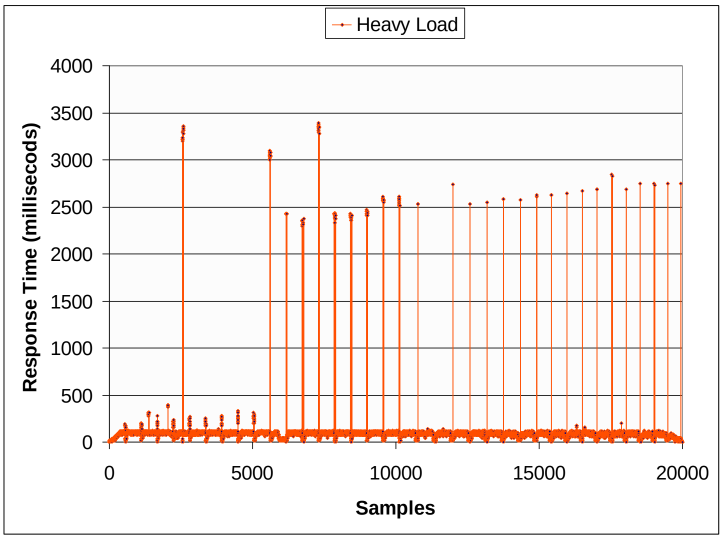
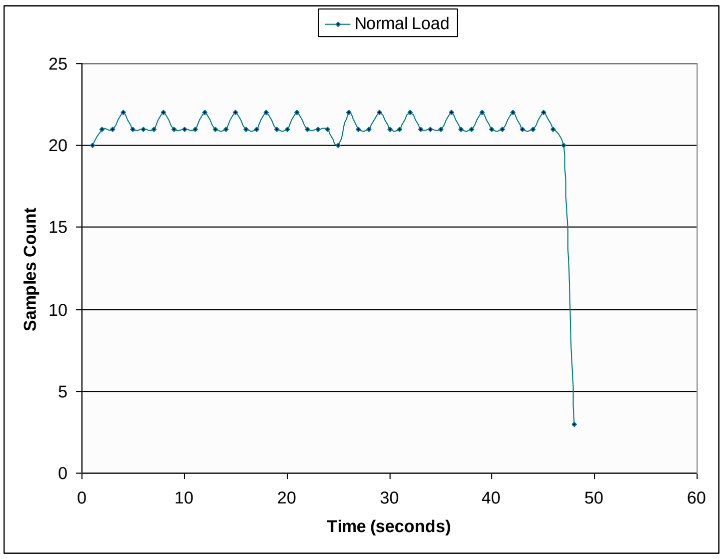
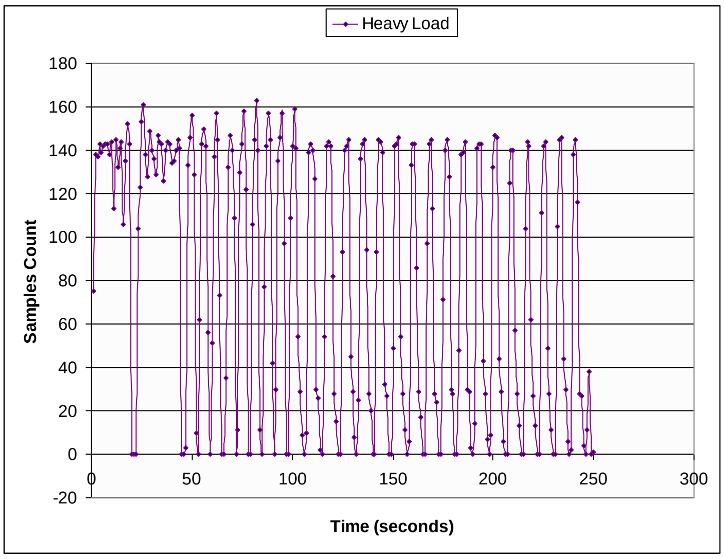
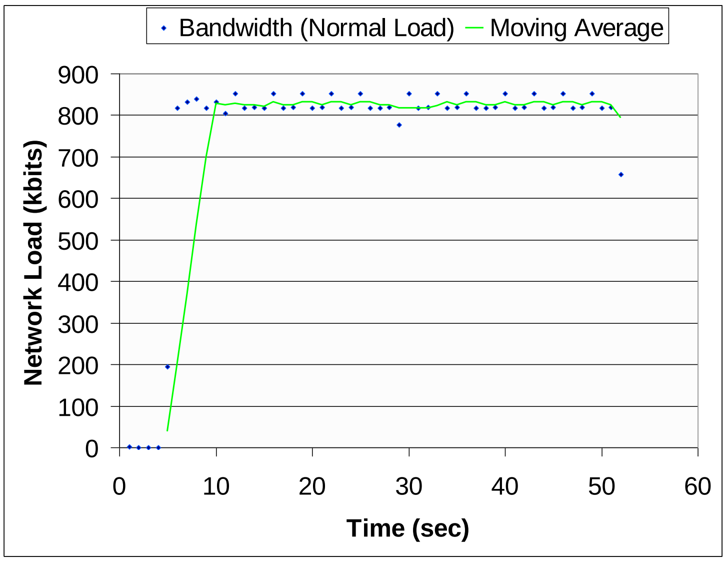
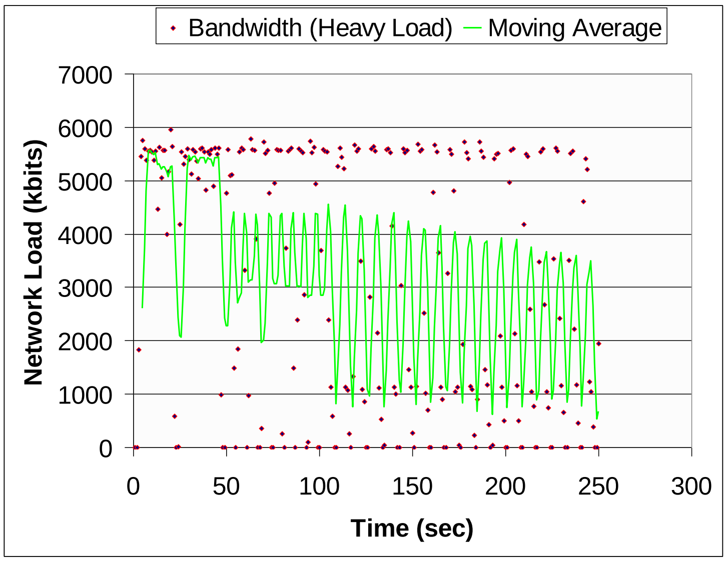
In a view of fault tolerance and reliability, the tests showed that even for the heavy load case, all the data was correctly delivered with no corrupted messages. Considering that, there were more than 20000 cycles, this result may be assumed as satisfactory.

5.2.2.2 SYSTEM CONTROL AGENT

The following table points out the average and maximum values in sets of measured parameters:

Category	**Average**	**Maximum**
Response time (in milliseconds):		
Normal Load (1000 cycles)	0,155	78
Heavy Load (20000 cycles)	105,14	3391
Throughput (samples/second):		
Normal Load (1000 cycles)	20,83	22
Heavy Load (20000 cycles)	80	163
Network Load (kbits/second):		
Normal Load (1000 cycles)	743,78	852
Heavy Load (20000 cycles)	3110,83	5956,68
CPU load (in %):		
Normal Load (1000 cycles)	16,43	17,9
Heavy Load (20000 cycles)	84,14	99,9

The following plots depict performance measurements for the System Control Agent.

Normal Load	Heavy Load
	
	
	

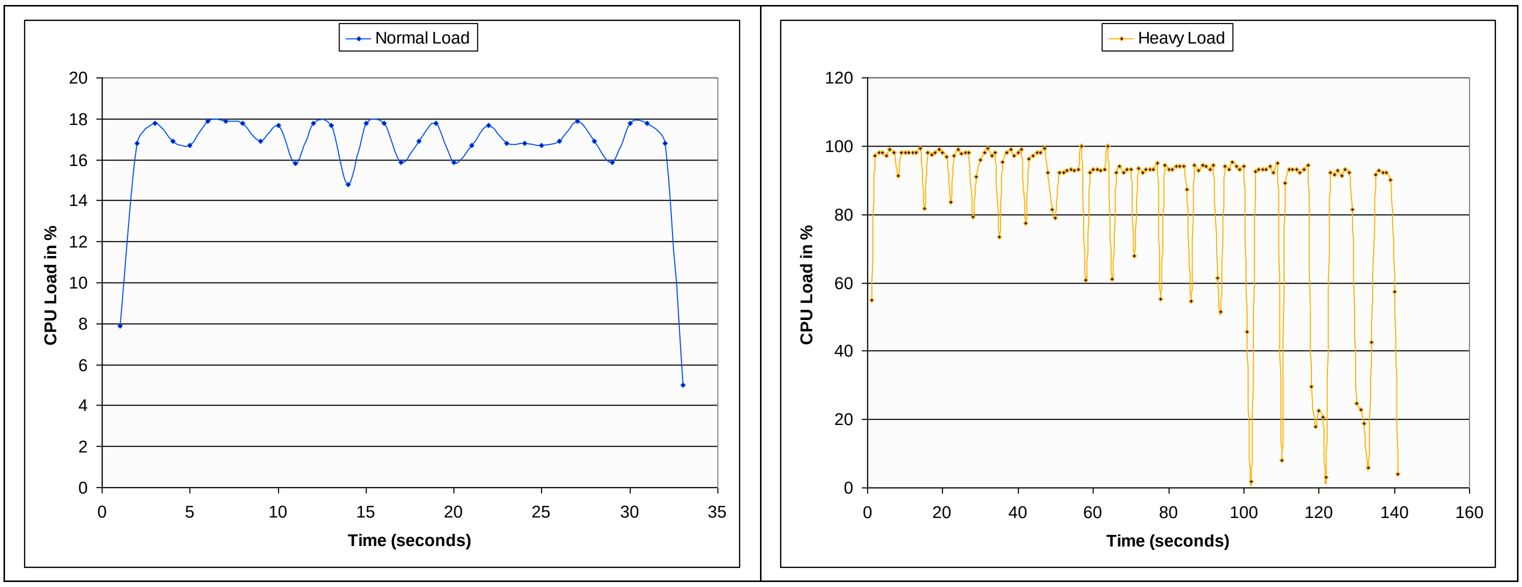


Figure 65. System Control Agent – Performance Measurements

This series of measurements showed less impressive results than for the System Monitoring Agent in regard to the response time. The average response time was no longer, than 80 milliseconds for the normal load case, while in the heavy load case there were many peaks of more than 3 seconds. However, considering a real use-case with no more than two requests per second, the response time seems be acceptable.

As for the throughput and network load, the results are comparable with the ones of the System Monitoring Agent. The overhead could not change and it is equal to 38,89 kbits. The small difference between the values is due to some fluctuations, which do not change the overall situation.

The CPU load in these tests is greater than for the System Monitoring Agent and is close to 100%. This means that the heavy load case is a critical one and it is very close to the satisfactory limit. One can note that although both the System Monitoring and Control Agents are based on the C++ adapter, there is a considerable difference in the CPU usage in the heavy load case. This can be explained by differences in internal structure of the agents, e.g. using of forked process for handling of concurrent requests.

In the supervision solution for the DIS-RVM system, it is planned to use both the System Monitoring and System Control Agents on each machine. Taking into account that there will be no more than two messages per second for each agent, the CPU load parameters can be assumed as acceptable.

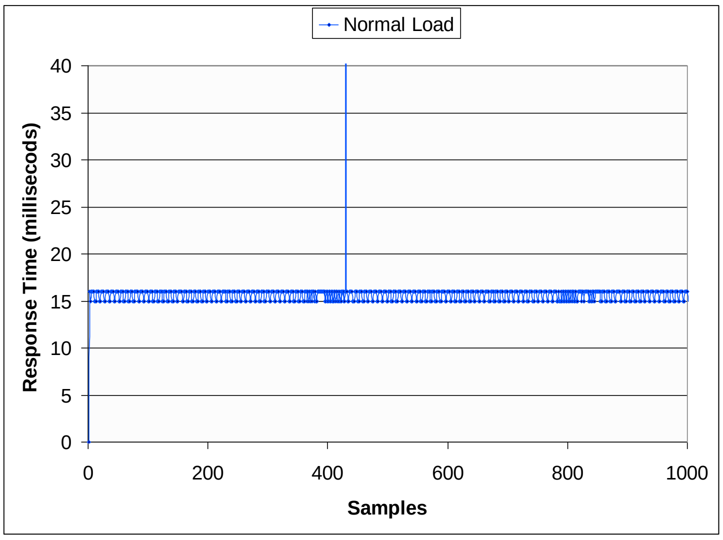
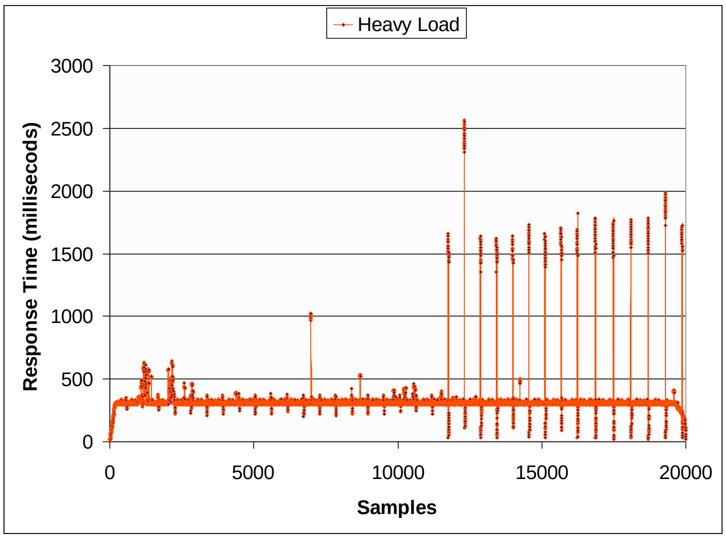
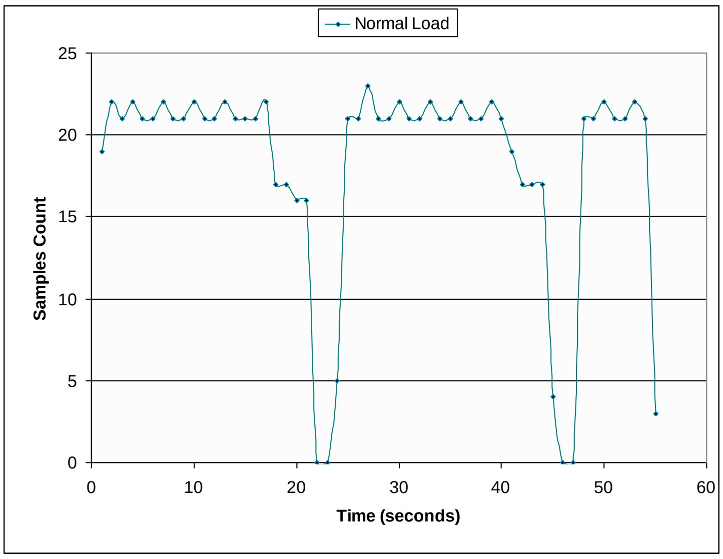
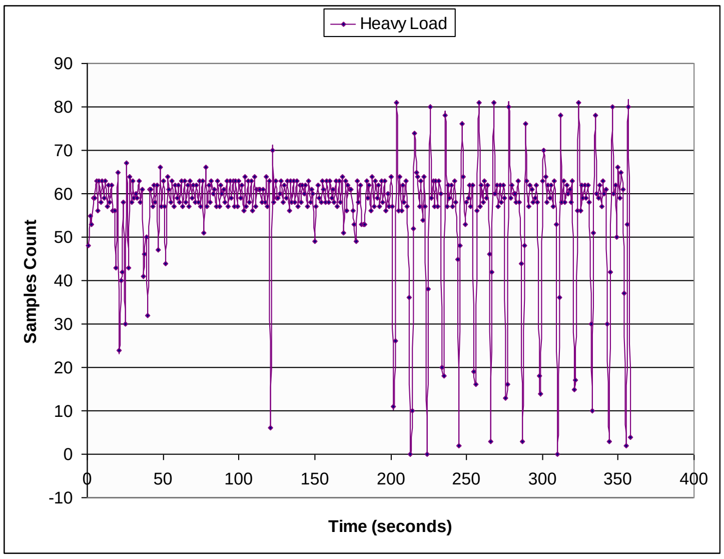
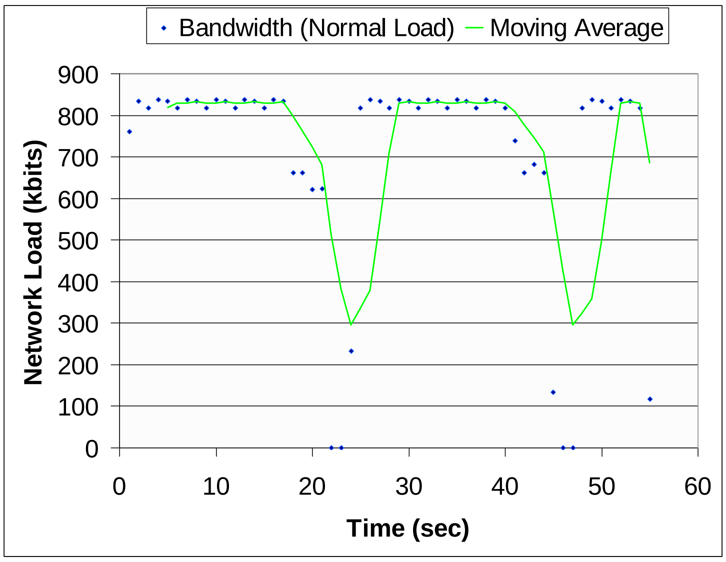
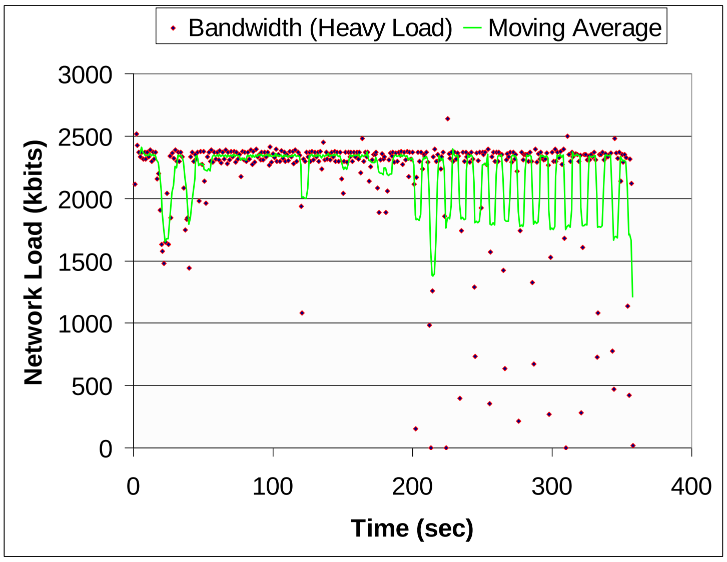
Considering fault tolerance and reliability, even for the cases, the load is close to the limit, the agent manages to process the communication, since there were no corrupted or dropped messages. This proves reliability of the concept implementation.

5.2.2.3 MÄK RTI MONITORING AGENT

The following table points out the average and maximum values in sets of measured parameters:

Category	**Average**	**Maximum**
Response time (in milliseconds):		
Normal Load (1000 cycles)	18,41	2797
Heavy Load (20000 cycles)	329,60	2563
Throughput (samples/second):		
Normal Load (1000 cycles)	18,18	23
Heavy Load (20000 cycles)	55,86	81
Network Load (kbits/second):		
Normal Load (1000 cycles)	707,12	838,02
Heavy Load (20000 cycles)	2172,69	2637,14
CPU load (in %):		
Normal Load (1000 cycles)	34,16	37,8
Heavy Load (20000 cycles)	67,32	92,3

The following plots depict performance measurements for the MÄK RTI Agent.

Normal Load	**Heavy Load**
	
	
	

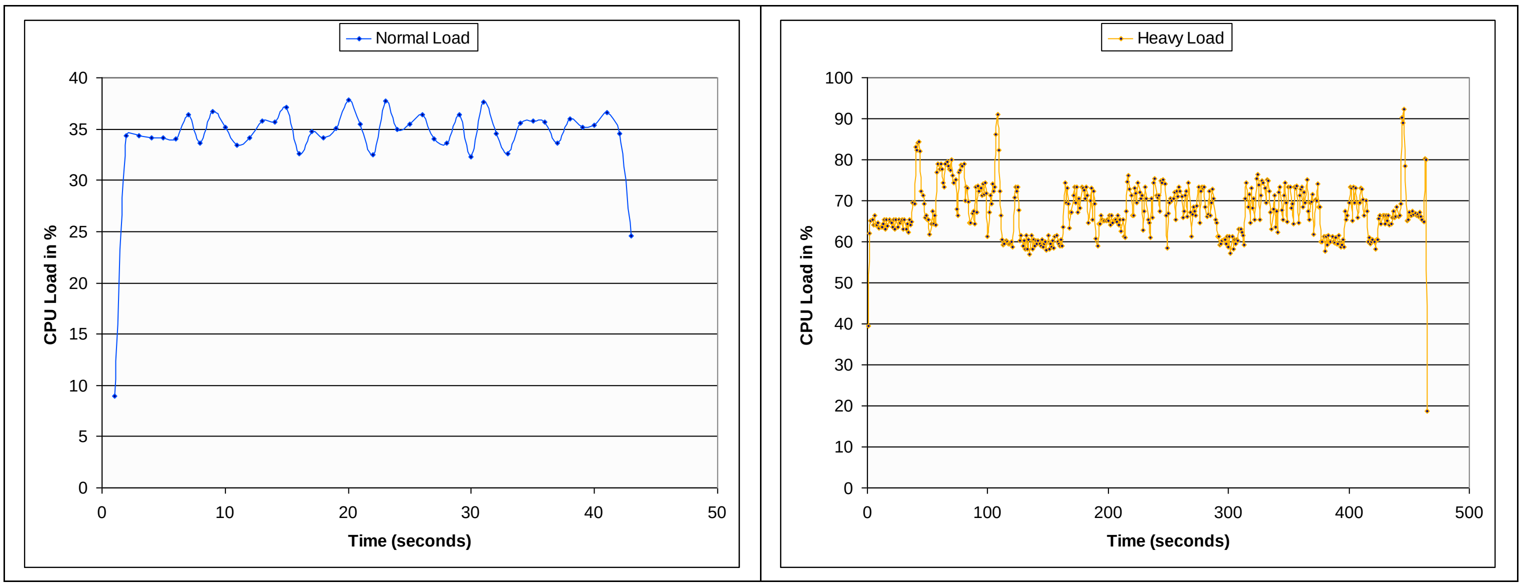


Figure 66. MÄK RTI Monitoring Agent – Performance Measurements

The MÄK RTI Monitoring Agent is more complex than the previous agents. It has two independent modules that are interconnected by means of sockets. In addition, it uses multithreading to manage communication streams between the modules. These implementation details explain the difference of the results for the response time measurement in the normal load case (18 milliseconds in average). Nevertheless, in the heavy load case, the response time is quite similar to the above-provided results.

The throughput is less for this agent than for the other ones, i.e. completion of the 20000 requests/responses took up to three times more than for the other agents. This means that the complexity of the internal structure highly affects productivity of message handling.

Considering the CPU load, the results are similar to results for the System Control Agent. In the heavy load case, since there was more time for message handling, the CPU load was rather less than for the System Control Agent. However, it is still not comparable with the results for the System Monitoring Agent.

The fault tolerance and reliability characteristics are similar to ones of the other agents, since all the cycles were successfully completed and there were no broken or dropped messages.

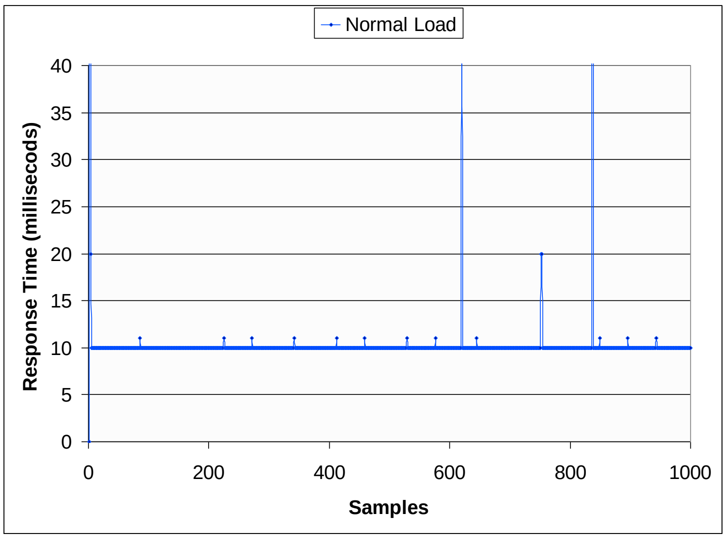
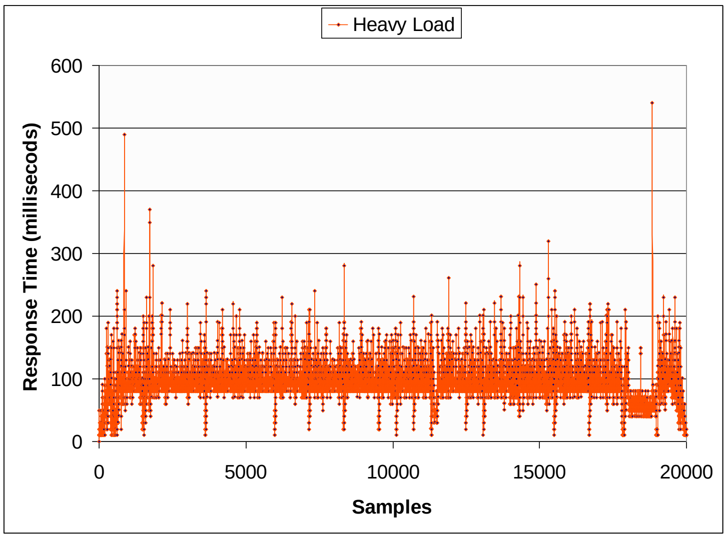
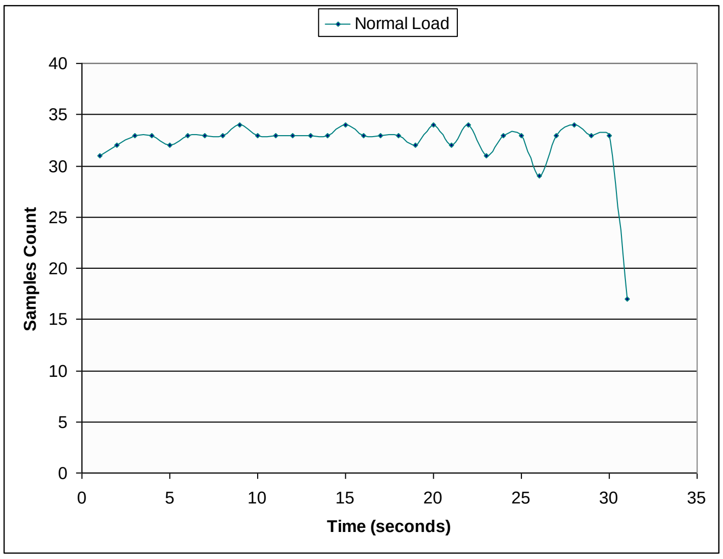
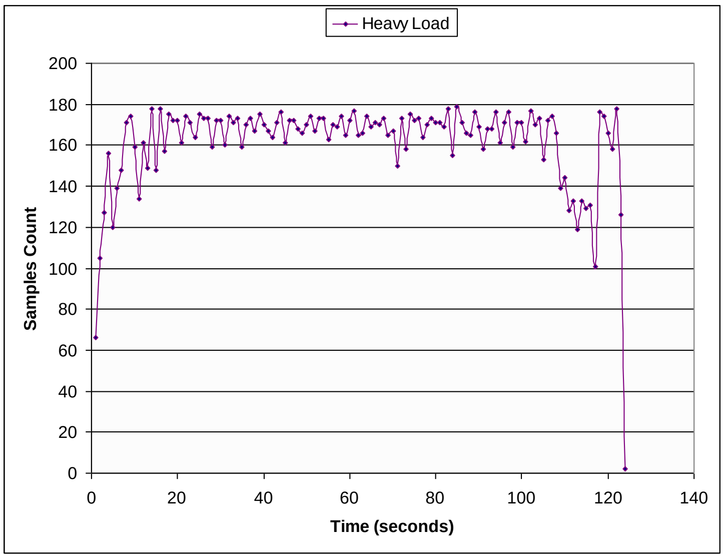
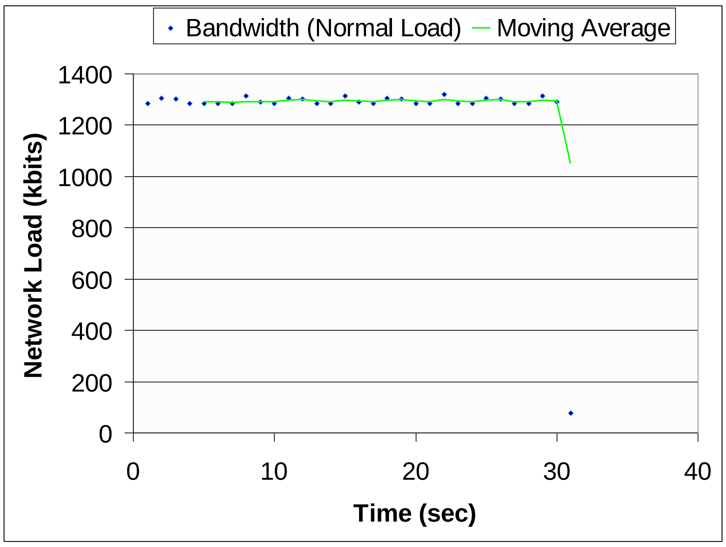
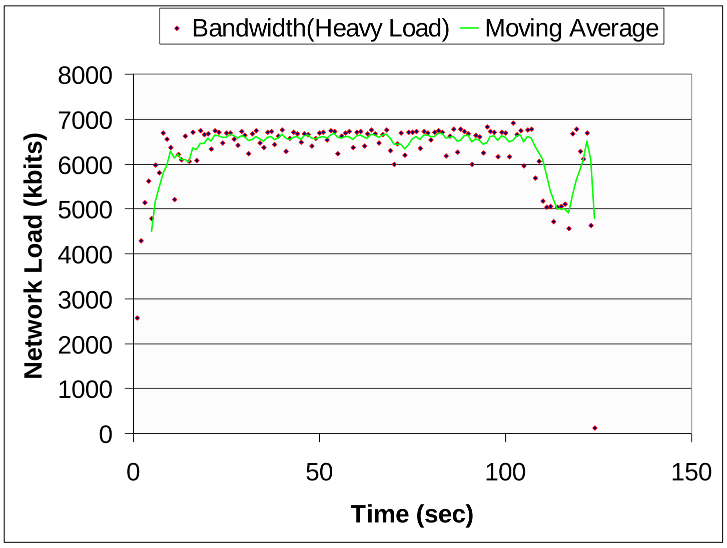
Equally to the other cases, network load parameters do not feet requirements for the DIS-RVM system. Nevertheless, other results can be assumed as acceptable for the DIS-RVM supervision, taking into account that there will be maximum two supervision requests for this agent.

5.2.2.4 DIS-RVM Monitoring Agent

The following table points out the average and maximum values in sets of measured parameters:

Category	**Average**	**Maximum**
Response time (in milliseconds):		
Normal Load (1000 cycles)	10,35	150
Heavy Load (20000 cycles)	88,86	540
Throughput (samples/second):		
Normal Load (1000 cycles)	31,25	34
Heavy Load (20000 cycles)	161,29	179
Network Load (kbits/second):		
Normal Load (1000 cycles)	1254,32	1318,7
Heavy Load (20000 cycles)	6271,43	69,0518
CPU load (in %):		
Normal Load (1000 cycles)	18,59	35,6
Heavy Load (20000 cycles)	92,43	99,9

The following plots depict performance measurements for the DIS-RVM Agent.

Normal Load	Heavy Load
	
	
	

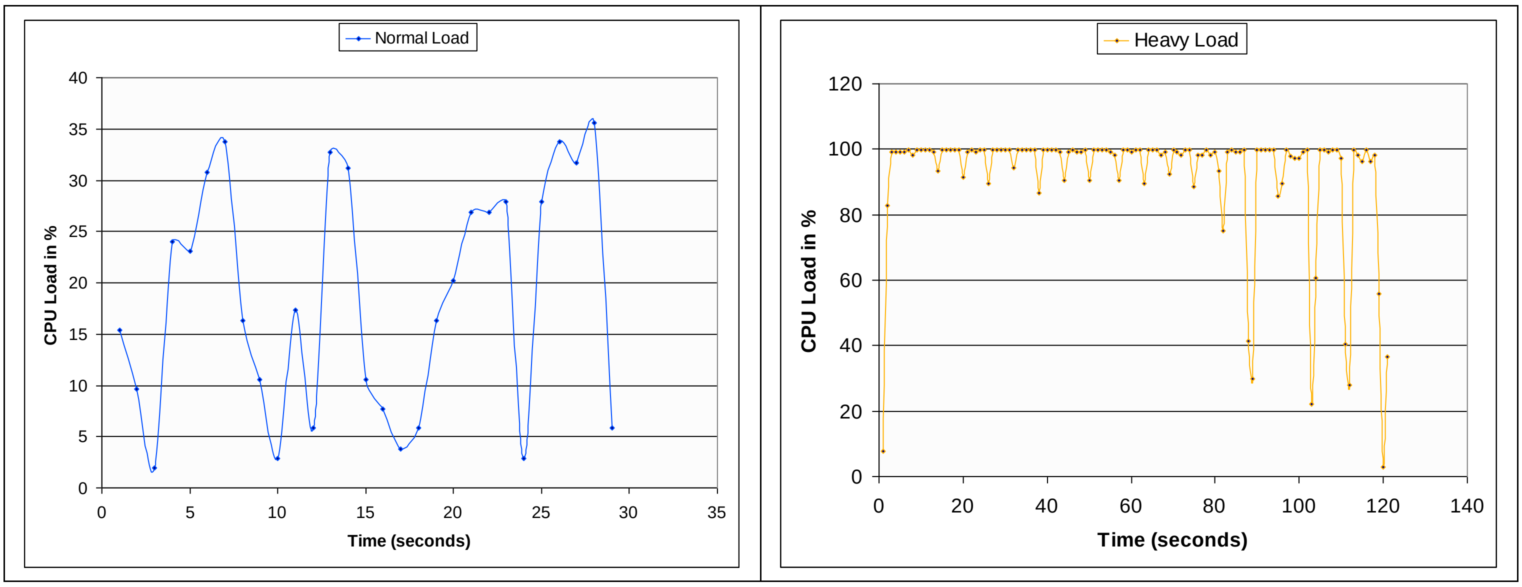


Figure 67. DIS-RVM Monitoring Agent – Performance Measurements

Internal structure of the DIS-RVM Monitoring Agent is much simpler than of the MÄK RTI Monitoring Agent. It uses a shared memory mechanism for communication with the DIS-RVM system. Consequently, the results are better than for the MÄK RTI Monitoring Agent. Although the response time for the normal case is close to the one of the MÄK RTI Monitoring Agent, the responsiveness in the heavy load case is quite close to the one of the System Monitoring Agent (100-200 milliseconds in average).

Besides, the throughput is again about 30/160 messages per second, which is very similar to the measurements for the system agents.

The CPU load is also comparable to the results of the System Control Agent. It is not greater than 40% for the normal load case and it is close to 100% for the heavy load.

In the same way, the reliability and fault tolerance were reconfirmed.

5.2.3 Conclusions

The performance tests showed limits of the proposed approach. The average response time varies up to 30 milliseconds, for the normal load case, while in the heavy load case, it does not exceed 350 milliseconds. Meanwhile for the heavy load, the 3 second peaks in response time are quite often. That means that message-handling engine was often overloaded in this case. On the other hand, in the normal load case, the responsiveness is comparable to middleware based on binary protocols (CORBA, JAVA RMI).

Considering the CPU load tests, 150 requests per second charge heavily a CPU and can block a supervised system. Therefore, heavily loaded components like supervision consoles are better to deploy in separate hosts.

In the supervision solution for the DIS-RVM system, it is planned to use two-three monitoring agents per host maximum, and, in average, three supervision operations per agent. Thus, considering one-second periodic subscription, there will be, in general, six-nine outcoming messages per second for each host. In this way, this load is much less than the one in the “normal” load scenario and, thus, in regard to responsiveness and CPU load, it can be assumed that the developed prototype can be used for supervision of the Distributed Training scenario.

However, the network tests pointed out that the overhead is about 38 kbits per request/response message pair (calculated as a ratio of the average network load parameter to the average throughput), which is critical for applications with strict bandwidth requirements, like the DIS-RVM system. Hence, for the DIS-RVM system, the supervision approach can be applied only locally. As the result of this evaluation, the validation session for the Distributed Training was conducted in LAN and, in addition, involved remote hosts with a fast Internet access. One can refer to section 5.4.2 for more details. Besides, one can find the results of the measurements of an overall network load during Distributed Training validation sessions there.

The importance of bandwidth issues is diminished in LANs and broadband WAN. Besides, the test approach concerned only periodic monitoring, therefore the network load can be optimised using event based subscription mechanism. Moreover, the situation can be improved by means of intelligent summarisers that locally collect the monitoring information and eventually send reports or alarms. In addition, in critical cases one can consider to use supervision approaches based on a binary protocol and to use the proposed concept as a gateway for communication with third party frameworks.

It is worth mentioning that even for the heavy load case, there were no corrupted or dropped messages during all the tests (more than 80000 cycles). This undoubtedly confirms reliability and fault tolerance of the proposed concept.

In the next section a description of the user validation process starts with the Preliminary Design Review scenario.

5.3 PRELIMINARY DESIGN REVIEW SCENARIO

As it was mentioned above, this scenario deals with a Collaborative Engineering system. In this case, an end-user application scenario provides an important method to verify how the supervision concept does actually meet the user expectations and needs. In particular, the PDR scenario should check the middleware capabilities in a widely distributed application. Besides, the end-users will evaluate the following aspects:

- **Comprehensiveness:** an ability to simultaneously supervise multiple parameters – operating systems, network and applications;
- **Interoperability:** an ability of heterogeneous components to cooperate among themselves. In this case, it is an ability of the C++ adapter agents to cooperate with Java based consoles;
- **Portability:** an ability of the agents to run under different operating systems;
- **Ease of deployment**.

The end-user validation activities have been organised in the context of the Preliminary Design Review (PDR) scenario. The PDR scenario is based on real processes handled in the space domain called program reviews. Descriptions of these processes and of the real projects (for example, the Automated Transfer Vehicle) PDR applies on are given in the Introduction and Prototype sections.

The following sections describe the validation activities in detail.

5.3.1 SCENARIO DESCRIPTION

The developed supervision system provides functionality to enhance the existing PDR application to:

- Supervise distributed PDR system including: EDB server and clients, groupware, logs, etc.
- Supervise PDR sessions live including: user management, session logs, etc.
- Supervise the PDR application execution environment including: operating systems and network accessibility of the involved hosts.

The validation scenario verifies the requirements by means of scenario rehearsals involving different groups of users:

Actors:

PDR scenario has the following actors representing different entities in a PDR process:

- The Reviewers - their role is to review RIDs (Review Item Discrepancy), to recommend corrective actions, etc.;

- The Review Manager - he will invite the participants for a review and supervise the session in terms of document management during the session (supervision of a review session);
- The Application Administrator - he is in charge of all PDR application and GroupWare tools on application level. He uses the supervision system to retrieve specific information about PDR application. In case of a system problem, he reports it to the System Administrator;
- The System Administrator - he is in charge of the operating system level of the entire system. It includes system installation and maintenance. He uses the supervision system to determine and solve problems.

The **RID Review** use-case was selected for the PDR scenario. During this use-case, Reviewers comment on documents according to the issued RIDs using PDR application and discuss modifications using groupware tools. Meanwhile, administrators monitor and maintain PDR system and groupware tools.

These actions are outlined in the following table in detail.

PDR Process Actions:	**Administrators Actions:**
1. Establish video conference session. The reviewers arrange a video conference session and try to join the conference server.	The administrators check the network status, connectivity and servers status using supervision system. In case of emergency, they try to recover the PDR Process.
2. Video conference session. The reviewers discuss agenda of the meeting.	The administrators check the network status, connectivity and servers status using supervision system. In case of emergency, they try to recover the PDR Process.
3. EDB session. The reviewers enter EDB system and chose a document to review. After the RID discussion, they leave their comments in the system.	The administrators check the network status, connectivity and servers status using supervision system. Application administrator verifies permanently the EDB system status.

5.3.2 VALIDATION PROCESS

The validation sessions involved all GeneSyS partners from France, Germany, Hungary and Russia. The Figure 68 shows the PDR and supervision elements repartition.

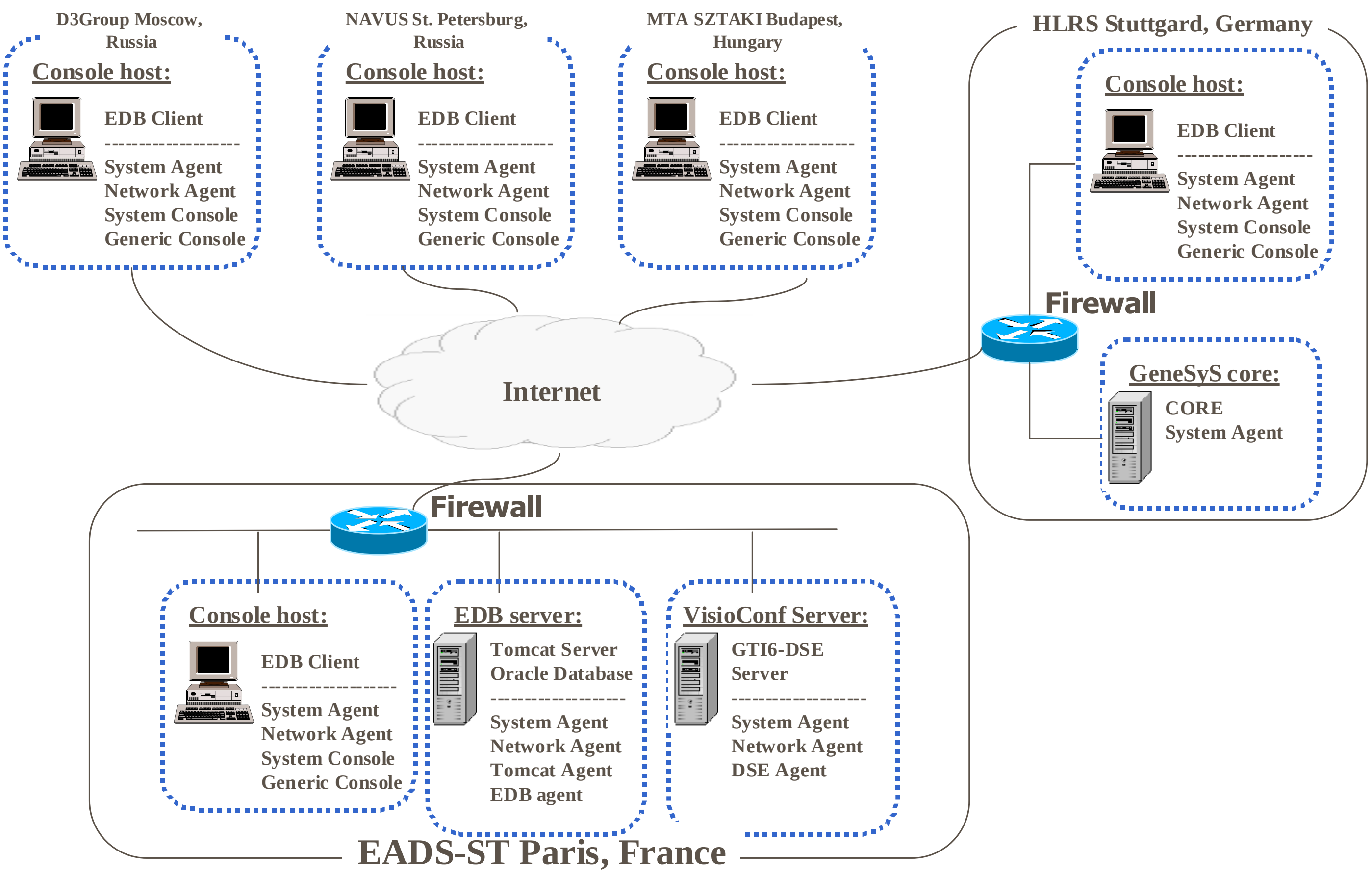


Figure 68. PDR Scenario Validation Platform

The participants at each platform played the scenario several times according to the plan described earlier. In addition, in order to check the supervision system reaction, the reviewers inserted some failures, while administrators tried to detect them. These failures included:

- CPU, Memory, Disk overload;
- network overload, connection break;
- improper video tool settings;
- EDB server access violation attempts, faulty operations with the documents.

All users reflected their observations in the evaluation forms for post-analyses.

5.3.3 Results

The PDR scenario illustrated the applicability of the proposed middleware to the collaborative systems supervision. The validation activities and user evaluation forms showed the following situation concerning the compliance of the supervision system:

Requirement:	**Validity is tested by:**
Interoperability/Portability	The System Monitoring Console, the EDB, Tomcat, GTI6-DSE,RTT and SNMP agents are Java based, while System agents are native applications, built on C++. In addition, the

	supervision system is deployed on hosts using Windows, Linux operating system. These components working together prove interoperability and portability of the proposed middleware.
Flexibility	The agents use complex messages that contain different kind of data in different forms (complex parameters, lists, etc.). The communication mechanism supports Query/Response and Publish Subscribe for periodic events. These aspects show certain flexibility of the supervision middleware.
Reliability	Users imitated fault simulations that were successfully detected. However, reaction time was rather long due to supervision system settings.
Ease of deployment	The components are easy to deploy. However, deploying and maintaining of tens of the components are rather time-consuming tasks.

The following section elaborates the applicability in another industrial context – distributed interactive simulations.

5.4 DISTRIBUTED TRAINING SCENARIO

As it was mentioned earlier, this scenario deals with a distributed interactive simulation for training of astronauts and ground controllers. In this case, an end-user application scenario provides an important method to verify how the supervision concept does actually meet the user expectations and needs. In particular, the Distributed Training scenario is intended to verify the middleware capabilities in systems with strict responsiveness constraints. Finally, the end-users validate the following aspects at a new level:

- **Comprehensiveness:** an ability to simultaneously supervise multiple parameters – operating systems, network and applications;
- **Interoperability:** an ability of heterogeneous components to cooperate among themselves. In this case, it is an ability of the C++ adapter agents to cooperate with Java based consoles;
- **Portability:** an ability of the same agents to run under different operating systems. In the scenario the System Control agent for Linux operating system, which is C++ adapter based, was also successfully used for the Windows operating system;
- **Control:** The scenario introduces an ability of agents to react, while the PDR scenario involved supervision system for monitoring only;
- **Simple Intelligence:** A summarising view concept, presented in the Prototype section, implements a simple intelligence solution that highlights a system health status and allows a simple error detection;
- **Flexibility:** The supervision system operation involved different types of data, which were successfully encoded according to the architectural requirements.

The end-user validation activities have been organised in the context of the Distributed Training scenario and are elaborated in the following sections in detail.

5.4.1 SCENARIO DESCRIPTION

The validation process involves different groups of actors:

- **The trainees:** they are plugged to their simulation application through their specific GUI: for example, for the astronauts, it is the International Space Station control panel, while for the ground controllers it is the Flight Controller Position Workstation from the Control Room;
- **The instructors:** they interact together to select and run a given exercise. For instance, they introduce a contingency situation to verify that the trainees recognise the event and timely react to it by applying a proper procedure;
- **The technical operators:** they are in charge of controlling their local infrastructure as well as the network access. They should constantly monitor it and immediately react to prevent or correct a contingency situation. For these purposes, the technical operators use the developed supervision system.

The validation scenario is divided into 3 different phases that are described below.

Simulator Start Up

During this phase, technical operators perform the following tasks:

- **System configuration:** All the simulation hosts should have NTP service (process) running, NTP configuration should be properly set-up (single NTP server should be used as a time reference on all the simulation hosts)
- **RTI Configuration and Startup:** RTI shall be configured by means of:
  - RTI Initialisation Data File (RID file). Such a file should be identical on all the simulation hosts.
  - Environment variables to be set depending on installation of RTI components on a particular host.

  In addition, RTI should be started on a pre-defined host before a simulation session.
- **Federation Configuration:** This configuration should be performed by putting an appropriate Federation Execution Data (FED) file to the predefined directories on the simulation hosts.
- **Application Configuration and Start-up:** DIS-RVM uses a certain number (up to 20) of configuration files and most of them should be identical on all the simulation hosts. Some of those files are specific for each simulation (training) session and have to be properly set-up each time a new session is scheduled. Besides, the federates-simulators should be started in a particular order.

The technical operators use DIS-RVM console to accomplish the DIS-RVM system start-up and to check overall system health status. When the system reports a “ready” status, the operators launch the simulation sessions for the instructors and the trainees.

Simulation Session

During this phase, the trainees play the Rendez-Vous and Docking mission scenario performing ATV approach and docking manoeuvres, while the Instructor inserts contingencies to the ATV and ISS federates behaviour.

The technical operators monitor system health status, detect system failures and try to recover them. To check this functionality, users emulate the following system failures:

- System Failures: CPU, Memory, Disk overloads;
- Network Failures: Connection break;
- Application failure: termination of the Federates.

Closedown and Post-analysis

Upon completion of a simulation session, technical operators should perform the following tasks:

- Shutdown DIS-RVM federates
- Shutdown RTI
- Retrieve (over network) Simulation Data Recorder files and log files of each federate for further post-analysis by training instructors.

These tasks are performed by the operators via the DIS-RVM console.

5.4.2 VALIDATION PROCESS

The DIS-RVM software and supervision system were deployed in the EADS Space Transportation premises, while Control Host and the Public Core (Directory Server) were accessed via the Internet. The repartition of the components is depicted in Figure 69.

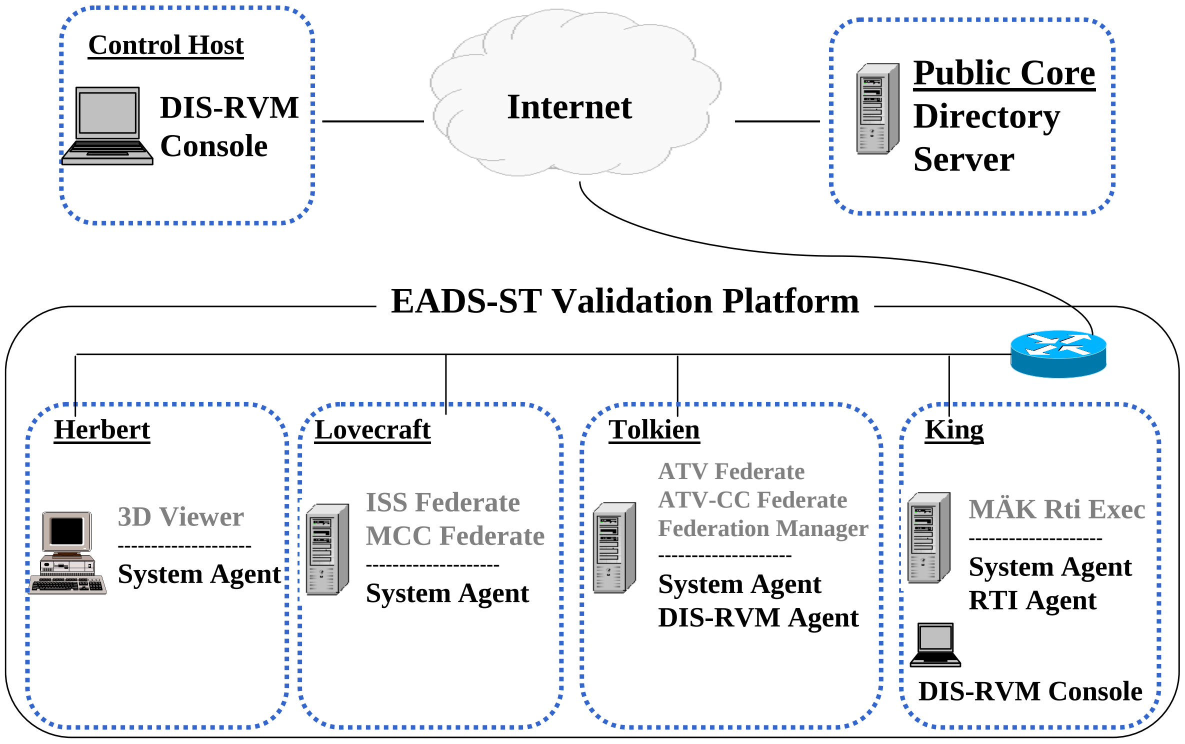


Figure 69. Distributed Training Scenario Validation Platform

Although the simulation process and training were conducted locally, the supervision was held both locally at EADS-ST premises and remotely accessing the validation platform via the Internet.

In addition to stand-alone performance tests, the network load was measured during the scenario rehearsals. The results of one of these tests can be found in Figure 70.

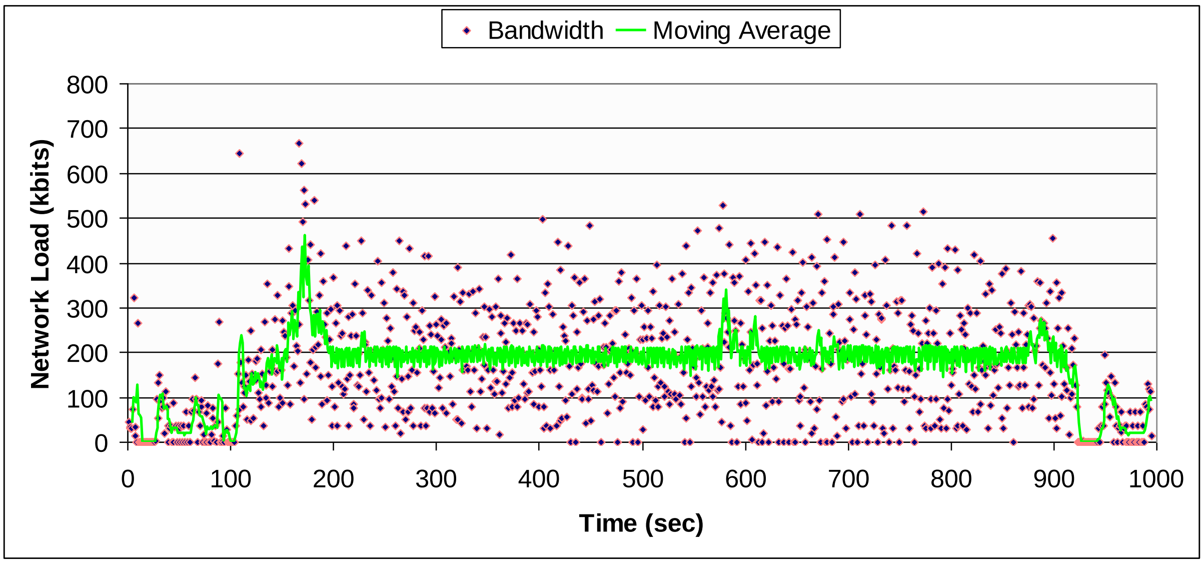


Figure 70. Distributed Training Scenario Network Load Measurement

The average network load was about 200kbit/s, which does not actually meet the requirements of the DIS-RVM system. This result is rather predictable according to the results of the performance tests. Most of the traffic consumption concerned periodic monitoring of the system, middleware and application parameters. Therefore, one can consider to use more intelligent summarising and event based monitoring. This would visibly diminish the network load.

Nevertheless, even 200kbit/s seems to be acceptable consumption for a wide range of application, since nowadays 512kbit/s is a usual bandwidth guaranteed by Internet Service Providers. 2Mbits/s and a larger bandwidth are available for a reasonable price and cannot be viewed as extraordinary. Telecom operators provide faster and faster Internet services and thus the outlined consumption issues will not be a stop factor for future distributed systems.

In addition, the above-mentioned consumption is quite acceptable in LANs. Hence, one of the options could be to locally deploy the proposed supervision approach and to diminish remote supervision by event-based summary and corrective commands.

As for the user validation sessions, the scenario rehearsals were performed in the same way as for the PDR scenario, i.e. different users played the all three phases of the scenario and inserted some faults that were successfully detected. Remarks of the actors were collected in evaluation forms for a post-analysis.

5.4.3 RESULTS

In spite of the above-mentioned consumption issues, the Distributed Training scenario showed applicability of the proposed supervision concept in another context of distributed applications – HLA compliant Distributed Interactive Simulations.

In addition, this scenario evolved performance aspects of the supervision approach, showing applicability limits. Agents could handle about 30 supervision requests per second without overcharging the hosting operating system. Even with 150 requests per second, there were no broken or missing messages, though the response time decreased considerably, as well as CPU load increased. This undoubtedly proves fault tolerance and reliability of the developed solution.

Considering the end-user validation, after analysis of the validation sessions and evaluation forms, the following requirements can be assumed as validated from a user's point of view:

Requirement:	**Validity is tested by:**
Interoperability/Portability	DIS-RVM Console is Java based, while all agents are native applications, built on C++. In addition, the supervision system is deployed on hosts using Windows, Linux operating system. This all proves interoperability and portability of the proposed middleware regarding programming languages and operating systems.
Flexibility	The agents use complex messages that contain different kinds of data in different forms. The communication mechanism supports Query/Response and Publish/Subscribe for periodic and asynchronous events.
Intelligence	The DIS-RVM console implemented special GUI that provided summarising view for status evaluation of the system health.
Reliability	Users imitated fault simulations that were successfully detected. However, reaction time was rather long due to

	supervision system settings.
Ease of deployment	The components are easy to deploy. However, deploying and maintaining tens of the components are rather time-consuming tasks, because of multiple configuration files to be managed.

The distributed supervision system relies on network services. However, the validation sessions showed that the detection of a connection break took a lot of time. This fact is an implementation issue, which could be identified only during user validation. It concerns the underlying Web Services middleware, which is based on SOAP-RPC protocol over HTTP. This protocol is stateless, in other words, a link is brought up every time a message is sent. Thus, a network connection failure can be detected only by sending periodically "ping" messages. Therefore, the reaction time cannot be greater than the period of ping messages. Consequently, to manage this issue, a solution can consist in involving network agents that permanently maintain connectivity and detect abnormal termination of a link.

The following section summarises the overall evaluation results.

5.5 CONCLUSIONS

The evaluation activities contained three phases representing different points of view: development lessons learned, performance tests and validation by end-users. These activities considered all the requirements for the supervision middleware as outlined below:

- **Comprehensiveness**: the developed solutions provided supervision on all the levels, i.e. system, network, middleware and application;
- **Interoperability**: during development interoperability issues were successfully solved. This resulted in adapters for C++, Java and PHP that will be reused for a further development;
- **Flexibility**: the supervision solutions required encoding of different data structures in supervision messages. The XML permitted to effectively manage this task. Besides, various communication mechanisms were implemented, i.e. single query/response, periodic notification, event-based notification;
- **Support of Intelligence**: A simple intelligence solution was implemented to demonstrate the flexibility. The summarising concept involved monitoring information evaluation at a delegate level and generic visualisation on a supervisor level. Although, it required additional specialised fields in each message, this confirms comprehensiveness and flexibility of the proposed approach;
- **Reliability and fault tolerance**: The performance tests verified the ability of the agents to work under a stress load. The supervision system behaved predictably and no messages were missing or corrupted. Moreover, end-users inserted contingencies during validation sessions, which were successfully detected;
- **Ease of development**: several adapters were developed for connection with the supervision framework. These adapters provide a paradigm for fast agent development and help to avoid interoperability issues. Besides, existing agents (freely available at [56]) may serve as examples;

- **Ease of deployment**: all the agents are stand-alone and do not require any additional libraries. Also, the agents can be deployed in web and application servers as web applications. However, deploying and configuring multiple agents may be a difficult task, which is a subject to further development.

The concept did not elaborate security aspects, because the Web-Service security is a well-developed domain with multiple implementations as described in the section 4.3.2.8. These specifications were constantly developing at the moment of prototyping, thus application level security implementation on an up-to-date level was blocked. Meanwhile, a transport level security (HTTPs) is a matter of configuration of Web Servers, hosting supervision agents.

The main added-value of this work is an investigation of the applicability limits of the concept. The performance elaborates this question, in regard to the responsiveness and system resources load. Although, the overhead of about 38 kbits is a usual situation in the XML world, this characteristic is unacceptable for the DIS-RVM system as it was specified. However, locally or in a broadband WAN, the supervision solution can still be applied and shows rather acceptable results, as it was demonstrated in section 5.4.2.

The performance tests consisted of two general cases: the normal load (1 request per 20ms) and the heavy load (20 concurrent requests per 20ms). The normal load case represents an estimation of a usual situation, while the heavy load case represents a stress test.

Considering the responsiveness in the normal load case, the most important result is that the prototype showed response time (1-20 ms) comparable to middleware based on binary protocols. The throughput in this case was about 20-30 handled requests per second depending on agent realisation. While CPU load varied from 8% for the System Monitoring Agent and up to 40% for other agents.

In the stress case, the response time varied up to three-second peaks, while throughput did not exceed 200 handled requests per second. The CPU load also depended on agent implementation. Although for the System Monitoring Agent it was rather acceptable (40%), for the other agents a CPU was heavily charged (100%) that inevitably affected the operating system. However, the most important result of this series of tests is that, even under stress conditions, all the requests were successfully processed and there were no abnormal interruptions of the handling and no missing or broken messages. This explicitly proves reliability and fault tolerance of the prototype in regard to stress situations.

In the meantime, the validation by the end-users demonstrated the concept applicability to collaborative engineering and distributed interactive simulation domains from a user's point of view. The actors played the predefined scenarios and inserted some faults that were detected in time. They pointed to usability issues in the PDR scenario that were solved in the Distributed Training scenario. They also evaluated an impact of the supervision to an overall system responsiveness to be acceptable.

All this permits to assume that the Web Services are applicable to distributed supervision systems to solve a wide range of problems.

Chapter 6. CONCLUSIONS

This section provides a work summary and outlines directions of possible further researches.

6.1 SUMMARY

The first part of this document introduces the context – complex systems, distributed systems, system supervision. The industrial problems present two complex systems in the spacecraft manufacturing domain (PDR and DIS-RVM systems) that illustrate the definitions and provide motivation for the present research. In addition, these examples serve to specify requirements for the supervision middleware and give particular details. These requirements were heavily used lately to evaluate the concept on the developed prototype.

The third chapter presents a state of the art in supervision technologies and frameworks. It identifies that the protocol based (SNMP) and interface/middleware based standards (JMX) determine the architectures of the major supervision frameworks (Unicenter TNG, OpenView, Tivoli etc.). This section also provides an analysis of the frameworks capabilities in regard to the identified requirements. In this way, the constraints of the available solutions motivate further research in the domain of the supervision middleware.

The fourth chapter starts with a comparison of middleware technologies. As a result, this analysis permits to assume that a Web Services based solution could provide advantages of the protocol based and middleware based frameworks. Therefore, it was decided to investigate the applicability of the Web Services for a supervision middleware.

The overview of the Web Services in the continuation shows flexibility of this technology and contributes to a better understanding of the architecture section. It outlines basic aspects of the XML technologies and provides a road map of Web Services security.

Further on the chapter presents the key value, i.e. a generic architecture of a supervision system based on the Web Services. It includes description of the basic components, operations, encoding of service information and a general example of a payload encoding. This architecture is the main innovation of this work and it was implemented in the GeneSyS project during the prototyping.

The fourth chapter ends with the description of the author’s contribution to the prototyping, which allowed verifying of the proposed concept in the context of real industrial problems. The developed supervision systems for collaborative engineering (PDR system) and distributed interactive simulations (DIS-RVM system) implement the proposed supervision approach. Besides, the simple approaches for agent collaboration and information summarising demonstrate the concept flexibility and its comprehensiveness. The C++ adapter effectively solves interoperability issues and provides means to rapidly develop new agents.

The fifth chapter presents another important added-value of the current work, i.e. evaluation of the concept applicability. The main results of the performance tests are that the average overhead of the proposed solution (service information encoding) is about 38 kbits per request/response message pair. Although this result is usual for the XML world, it clearly indicates that the supervision solution is not acceptable for the DIS-RVM system according to its specification (64kbps). However, as it showed lately, in local area networks and broadband wide area networks, the result is rather acceptable, since for the DIS-RVM system the average bandwidth consumption was about 200kbps. Moreover, the applied approach mostly relies on a periodic monitoring, while the intelligent event-based solution would significantly decrease the network load. Besides, the event-based approach has already been implemented in the MÄK RTI Monitoring Agent and, thus, it is available for further investigation.

In addition, the performance tests showed that under certain conditions (1 request per 20ms) the Web Service based solution shows the responsiveness (1 ms) comparable to middleware based on binary protocols. Moreover, even under the stress tests that charged CPU in 100%, the agents behaved correctly, though the response time decreased significantly. No corrupted or missing messages were detected. Hence, this proves reliability and fault tolerance in regard to a stress usage.

In the meantime, the end-user validation activities demonstrated concept usability. They were organised in validation scenarios making real actors evaluate middleware operation from a user's point of view. Besides, the system successfully resisted to contingencies that were inserted by actors, i.e. network connection breaks, operating system overload, application abnormal terminations and etc. The reviewers positively evaluated the proposed approach, which was noted in the validation forms.

All these actions confirmed the compliance with the requirements outlined in the introduction chapter. In this way, the performance tests permit to assume that the current approach is applicable for a wide range of problems, where flexibility and interoperability are of major interest.

6.2 TECHNOLOGICAL IMPACT

The current section elaborates industrial impact of the Web Services for the distributed system supervision.

Recently, several workgroups provided their specifications for the Web Services dealing with system supervision. These specifications include:

- OASIS MUWS - Management Using Web Services;
- OASIS-WSRF - WS-ResourceDescription, WS-Notification;
- WS-Management - WS-Eventing, WS-Enumeration from Microsoft;
- WSLA - Web Service Level Agreements.

The architectures proposed in these specifications are surprisingly similar to the concept architecture considered in this thesis and implemented in the GeneSyS project. Besides, the above-mentioned initiatives are still in the early stages, without any real implementation or a use-case, while the validation activities of the GeneSyS project resulted in supervision systems for:

- Collaborative Engineering – EADS-ST PDR System;
- Distributed Interactive Simulation – EADS-ST DIS-RVM System;
- Web Servers – MTA SZTAKI Dictionary.

In addition, the GeneSyS results are planned to be used in the following projects in scientific and industrial domains:

- EU IST FP6 projects ASPIC and BROADWAN;
- Collaborative robotics (University of Würzburg);
- MMOG middleware (Fraunhofer Gesellschaft);
- Mobile game platform (University of Turin);
- SMASH supercomputers based on FPGA devices;
- Tallisys CRUK project.

All this permits to assume that a considerable success of the GeneSyS project explicitly proves applicability of the proposed concept.

6.3 DIRECTIONS FOR FUTURE RESEARCHES

The work has raised multiple questions that can be shortly summarised in the following list:

- A common architecture for the middleware adapters should help to develop generic design patterns for supervision agents;
- Investigation of an efficient encoding of service information and payload would help to optimise network traffic;
- Supervision agents choreography would elaborate auto-discovery, auto-configuration capabilities and improve agents collaboration;
- Researches in peer-to-peer networks of supervision agents seem to be a logical continuation of the agents choreography;
- Further research in the intelligence for system supervision would permit to establish automatic recovery based on system reactions and trends;
- Collaboration with the standardisation groups like WBEM (CIM) and WSDM (OASIS) would move the concept towards standardisation.

6.4 AFTERWORD

According to SUN Microsystems, the computer is a network. According to Intel, the distributed systems move towards the peer-to-peer networks. According to Microsoft, the Web Services are a panacea. In any case, the distribution problems evolve the need of the system supervision.

Chapter 7. BIBLIOGRAPHY

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

[2] WordIQ dictionary http://www.wordiq.com

[3] Zhir Tari, Omar Bukhres. “Fundamentals of Distributed Object Systems: The CORBA Perspective”, John Wiley&Sons, Inc. 2001.

[4] ECSS Space Engineering, System Engineering (ECSS-E-10A). ESA-ESTEC Requirements & Standard Division, Noordwijk, The Netherlands.

[5] IEEE standard 1516: IEEE Standard for Modelling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules. 2000.

[6] IEEE standard 1516.2: HLA OMT -- High Level Architecture Object Model Template Spec. 2000

[7] High Level Architecture Federation Development and Execution Process (FEDEP) Model. Version 1.0. 6 September 1996

[8] RFC-792 – ICMP. http://rfc.net/rfc792.html

[9] Simple Network Management Protocol, http://www.snmp.org

[10] Sun Java web-site, http://java.sun.com

[11] IBM Tivoli, http://www 306.ibm.com/software/tivoli/

[12] HP OpenView, http://www.openview.hp.com/

[13] Nagios, http://www.nagios.org/

[14] Andrew Tanenbaum, “Distributed Operating Systems”, Prentice Hall, 1994, ISBN: 0132199084

[15] Gunter Rackl, “Monitoring and Managing Heterogeneous Middleware”, Ph.D. thesis, Technische Universitat Munichen, 2001, http://tumb1.biblio.tu-muenchen.de/publ/diss/in/2001/rackl.pdf

[16] I. Liabotis, O. Prnjat, L. Sacks, “Policy-Based Resource Management for Application Level Active Networks”, Second IEEE Latin American Network Operations and Management Symposium LANOMS 2001; Belo Horizonte, Brazil, August 2001

[17] O. Prnjat, T. Olukemi, I. Liabotis, L. Sacks, “Integrity and Security of the Application Level Active Networks”, IFIP Workshop on IP and ATM Traffic Management WATM’2001 and EUNICE’2001; Paris, France, September 2001

[18] Wokoma, I. Liabotis, O. Prnjat, L. Sacks, I. Marshall, “A Weakly Coupled Adaptive Gossip Protocol for Application Level Active Networks”, IEEE 3rd International Workshop on Policies for Distributed Systems and Networks - Policy 2002, Monterey, CA, USA, June 2002;

[19] I. Liabotis, O. Prnjat, L. Sacks; “Self-Organizing Resource Discovery Protocol for ALAN”, IFIP Fifth Annual International Working Conference on Active Networks (IWAN), December 10-12, 2003 at Kyoto Research Park, Kyoto, Japan

[20] A. A. Sebyala, T. Olukemi, L. Sacks, “Active Platform Security through Intrusion Detection Using Naïve Bayesian Network for Anomaly Detection”, London Communications Symposium, London, UK, 2002

[21] Wijngaards, N.J.E., Overeinder, B.J., Steen, M. van and Brazier, F.M.T., “Supporting Internet-Scale Multi-Agent Systems”, Data and Knowledge Engineering (2002), Vol. 41, Number 2-3, pp. 229-245

[22] Overeinder, B.J., Wijngaards, N.J.E., Steen, M. van and Brazier, F.M.T. (2002), “Multi-Agent Support for Internet-Scale Grid Management”, Proceedings of the AISB'02, Symposium on AI and Grid Computing, pp. 18-22, 2002

[23] Kun Yang, Alex Galis, Telma Mota, Stelios Gouveris, “Automated Management of IP Networks Through Policy and Mobile Agents”, Proceedings of the 4th International Workshop on Mobile Agents for Telecommunication Applications, 2002, ISBN:3-540-00021-6

[24] K. Yang, A. Galis, T. Mota, A. Michalas, “Mobile Agent Security Facility for Safe Configuration of IP Networks”, Second International Workshop on SECURITY OF MOBILE MULTIAGENT SYSTEMS (SEMAS-2002), Bologna, Italy, 2002

[25] T. Mota, S. Gouveris, G. Pavlou, A. Michalas, J. Psoroulas, “Quality of Service Management in IP Networks Using Mobile Agent Technology”, Mobile Agents for Telecommunication Applications : 4th International Workshop, MATA 2002, Barcelona, Spain, October 23-24, 2002

[26] P. Felber, X. Defago, P. Th. Eugster, A. Schiper, “Replicating CORBA objects: a marriage between active and passive replication”, Proceedings of the IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems II, 1999, ISBN:0-7923-8527-6

[27] R. Capobianchi, A. Coen-Porisini, D. Mandrioli, A. Morzenti, “A framework architecture for supervision and control systems”, ACM Computing Surveys (CSUR) archive, Volume 32 , Issue 1es, March 2000, ISSN:0360-0300

[28] F. Marotta, A. Morzenti, D. Mandrioli, “Modeling and Analyzing Real-Time CORBA and Supervision & Control Framework and Applications”, The 21st International Conference on Distributed Computing Systems, Mesa, AZ, USA, April 16 - 19, 2001

[29] The Open Group. The Open Group Portal to DCE. http://www.opengroup.org

[30] The Object Management Group. http://www.omg.org

[31] Microsoft Corporation. http://www.microsoft.com

[32] Foundation for Intelligent Physical Agent http://www.fipa.org

[33] XML Specification, http://www.w3.org/TR/REC-xml

[34] XML Design and Implementation, by Paul Spencer, Wrox Press Ltd., 1999

[35] XML Schema Standard, http://www.w3.org/TR/xmlschema-2/

[36] SAX Parser Specification, http://www.saxproject.org/

[37] DOM Parser Specification, http://www.w3.org/TR/DOM-Level-2-Core/

[38] World Wide Web Consortium specifications http://www.w3c.org and UDDI organisation http://www.uddi.org

[39] Agentlink project http://www.agentlink.org

[40] Web Services organisation http://www.webservices.org

[41] James Snell, Doug Tidwell, Pavel Kulchenko: "Programming Web Services with SOAP" O'Reilly&Associated, Inc. January 2002

[42] Heather Kreger: "Web Services Conceptual Architecture" IBM Software Group, May 2001. http://www-3.ibm.com/software/solutions/webservices/pdf/WSCA.pdf

[43] Luis Arguello, Alexander Vankov, Petr Chliaev, Vladimir Voloshinov, Valery Krivtsov, Oleg Estehin, Alexander Alyoshin, Alexander Vislotsky, Andrey Sadovykh : "Distributed Learning with Online Simulations for ISS Payload Training" ESA/ESTEC Conference, Noordwijk, The Netherlands, November 2002

[44] A. Sadovykh, S. Wesner, J-E Bohdanowicz: "GeneSyS: A Generic Architecture for Supervision of Distributed Applications" EuroWeb 2002 Conference Proceedings, pp. 38-42, Oxford, UK, December 2002

[45] Birrell, A.D. & Nelson, B.J. "Implementing Remote Procedure Calls." ACM Transactions on Computer Systems 2, 1 (February 1984): 39-59.

[46] RFC 2616, Hypertext Transfer Protocol - HTTP/1.1, June, 1999. http://www.normos.org/ietf/rfc/rfc2616.txt

[47] Simple Object Access Protocol specification, W3C, http://www.w3.org/TR/SOAP/

[48] Web Service Description Language, W3C, http://www.w3.org/TR/WSDL/

[49] UDDI Version 3.0. Published Specification, 19 July 2002. http://uddi.org/pubs/uddi-v3.00-published-20020719.htm

[50] UDDI Programmers API, http://www.uddi.com/pubs/ProgrammersAPI-V1.01-Open-20010327_2.pdf

[51] Apache Soap 2.3+ and Apache Axis Client Interoperability Results http://www.apache.org/~rubys/ApacheClientInterop.html

[52] Security in a Web Services World: A Proposed Architecture and Roadmap, 2002. http://www-106.ibm.com/developerworks/library/ws-secmap/

[53] MÄK RTI Reference Manual, Revision RTI-2.0.1-1-030530, 2003. http://www.mak.com

[54] Apache JMeter homepage, http://jakarta.apache.org/jmeter/

[55] Ethereal Network Protocol Analyser homepage, http://www.ethereal.com/

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

Chapter 8. LIST OF PUBLICATIONS

1. Luis Arguello, Alexander Vankov, Peter Shlyaev, Vladimir Voloshinov, Valery Krivtsov, Oleg Estehin, Alexander Alyoshin, Alexander Vislotsky and Andrey Sadovykh: “DISTRIBUTED LEARNING WITH ONLINE SIMULATIONS”, ESA/ESTEC Conference: 7th International Workshop on Simulation for European Space Programmes, Noordwijk, The Netherlands - November, 2002
2. A. Sadovykh, S. Wesner, J-E Bohdanowicz: "GeneSyS: A Generic Architecture for Supervision of Distributed Applications", EuroWeb 2002 Conference Proceedings, pp. 38-42, Oxford, UK, December 2002
3. Jean-Eric Bohdanowicz, Antoine Laydier, Peter Shlyaev, Alexander Vankov, Vladimir Voloshinov and Andrey Sadovykh: “GeneSyS Project: Supervision of Distributed Systems”, EuroSIW03: European Simulation Interoperability Workshop 2003, Stockholm, Sweden - June, 2003
4. Andrey Sadovykh, “Innovative Concept of Generic System Supervision”, DAIS FMOODS 2003 Conference: PhD Workshop, Paris, France - November, 2003
5. Jean-Eric Bohdanowicz, Antoine Laydier, Peter Shlyaev, Vladimir Voloshinov, Alexander Vankov and Andrey Sadovykh, “GeneSyS Project: Supervision of Distributed Training System for Astronauts and Ground Controllers (04E-SIW-023)”, EuroSIW04: European Simulation Interoperability Workshop 2004, Edinburgh, UK (Scotland) - June, 2004
6. Jean-Eric Bohdanowicz, Stefan Wesner, Laszlo Kovacs, Hendrik Heimer and Andrey Sadovykh, “THE PROBLEMATIC OF DISTRIBUTED SYSTEMS SUPERVISION - AN EXAMPLE : GENESYS”, IFIP WCC 2004 congress, Toulouse, France – August, 2004
7. J.E. Bohdanowicz, L. Kovacs, B. Pataki, A. Sadovykh and S. Wesner, “GeneSyS: Innovative Framework for Comprehensive Supervision in Multiple Domains”, IADIS WWW / Internet 2004 Conference, Madrid, Spain – October, 2004
8. J.-E. Bohdanowicz, L. Kovacs, B. Pataki, A. Sadovykh and S. Wesner, “On Distributed System Supervision - A Modern Approach: GeneSyS”, Network Control and Engineering for QoS, Security and Mobility, IFIP TC6 Conference, Palma de Mallorca, Spain - November, 2004

Chapter 9. ANNEXES

9.1 CORE WEB SERVICES DEFINITION

WSDL **Core.wsdl**

WSDL location:	**\XSD\Core.wsdl**
targetnamespace:	**http://core.namespace**

services	bindings	porttypes	messages
find	**find_SOAP_BINDING**	**find_PORT_TYPE**	**find_FAULT**
getSiteList	**getSiteList_SOAP_BINDING**	**getSiteList_PORT_TYPE**	**find_REQUEST**
register	**register_SOAP_BINDING**	**register_PORT_TYPE**	**find_RESPONSE**
unregister	**unregister_SOAP_BINDING**	**unregister_PORT_TYPE**	**getSiteList_FAULT**
updateRegistration	**updateRegistration_SOAP_BINDING**	**updateReqistration_PORT_TYPE**	**getSiteList_REQUEST**
			getSiteList_RESPONSE
			headerBlock
			register_REQUEST
			register_RESPONSE
			registration_FAULT
			unregister_REQUEST
			unregistration_FAULT
			updateRegistration_FAULT
			updateRegistration_REQUEST

service **register**

ports	**register_PORT**

binding	**y:register_SOAP_BINDING**
extensibility	**<soap:address location="No Target Adress"/>**

source
```
<service name="register">
  <port name="register_PORT" binding="y:register_SOAP_BINDING">
    <soap:address location="No Target Adress"/>
  </port>
</service>
```

service **unregister**

ports **unregister_PORT**

binding	**y:unregister_SOAP_BINDING**
extensibility	**<soap:address location="No Target Adress"/>**

source
```
<service name="unregister">
  <port name="unregister_PORT" binding="y:unregister_SOAP_BINDING">
    <soap:address location="No Target Adress"/>
  </port>
</service>
```

service **updateRegistration**

ports **updateRegistration_PORT**

binding	**y:updateRegistration_SOAP_BINDING**
extensibility	**<soap:address location="No Target Adress"/>**

source
```
<service name="updateRegistration">
  <port name="updateRegistration_PORT" binding="y:updateRegistration_SOAP_BINDING">
    <soap:address location="No Target Adress"/>
  </port>
</service>
```

service **getSiteList**

ports **getSiteList_PORT**

binding	**y:getSiteList_SOAP_BINDING**
extensibility	**<soap:address location="No Target Adress"/>**

source
```
<service name="getSiteList">
  <port name="getSiteList_PORT" binding="y:getSiteList_SOAP_BINDING">
```

```
    <soap:address location="No Target Adress"/>
  </port>
</service>
```

service **find**

ports	**find_PORT**
bin ding	**y:find_SOAP_BINDING**
exte nsibility	**<soap:address location="No Target Adress"/>**

source

```
<service name="find">
  <port name="find_PORT" binding="y:find_SOAP_BINDING">
    <soap:address location="No Target Adress"/>
  </port>
</service>
```

binding **register_SOAP_BINDING**

type	y:register_PORT_TYPE
extensibi lity	<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
operatio ns	**Register**
exte nsibility	
inp ut	**<soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/><soap:body parts="agentInfo" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>**
out put	**<soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/><soap:body parts="agentId" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>**
used by	Service **register** in Port **register_PORT**

source

```
<binding name="register_SOAP_BINDING" type="y:register_PORT_TYPE">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="Register">
    <input>
      <soap:header message="y:headerBlock" part="header" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      <soap:body parts="agentInfo" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </input>
    <output>
      <soap:header message="y:headerBlock" part="header" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      <soap:body parts="agentId" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </output>
    <fault name="registrationFault">
      <soap:header message="y:headerBlock" part="header" use="encoded"
```

```
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      <soap:fault name="registrationFault" use="literal" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </fault>
  </operation>
</binding>
```

binding **unregister_SOAP_BINDING**

type	**y:unregister_PORT_TYPE**
extensibility	<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
operations	**Unregister** **extensibility** **<soap:operation soapAction="urn:#Unregister"/>** **input** **<soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>** **output**
used by	Service **unregister** in Port **unregister_PORT**
source	

```
<binding name="unregister_SOAP_BINDING" type="y:unregister_PORT_TYPE">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="Unregister">
    <soap:operation soapAction="urn:#Unregister"/>
    <fault name="unregistrationFault">
      <soap:header message="y:headerBlock" part="header" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      <soap:fault name="unregisterFault" use="literal" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </fault>
    <soap:operation soapAction="urn:#Unregister"/>
    <input>
      <soap:header message="y:headerBlock" part="header" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </input>
  </operation>
</binding>
```

binding **updateRegistration_SOAP_BINDING**

type	**y:updateRegistration_PORT_TYPE**
extensibility	<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
operations	**UpdateRegistration** **extensibility** **<soap:operation soapAction="urn:#UpdateRegistration"/>** **input** **<soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/><soap:body parts="agentInfo" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>**

	output
used by	Service **updateRegistration** in Port **updateRegistration_PORT**

source

```
<binding name="updateRegistration_SOAP_BINDING" type="y:updateReqistration_PORT_TYPE">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="UpdateRegistration">
    <soap:operation soapAction="urn:#UpdateRegistration"/>
    <input>
      <soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      <soap:body parts="agentInfo" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </input>
    <soap:operation soapAction="urn:#UpdateRegistration"/>
    <fault name="updateRegistrationFault">
      <soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      <soap:fault name="updateRegistrationFault" use="literal" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </fault>
  </operation>
</binding>
```

binding **getSiteList_SOAP_BINDING**

type	**y:getSiteList_PORT_TYPE**	
extensibility	<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>	
operations	**GetSiteList**	
	extensibility	**<soap:operation soapAction="urn:#GetSiteList"/>**
	input	**<soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/><soap:body parts="ownerId" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>**
	output	**<soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/><soap:body parts="siteList" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>**
used by	Service **getSiteList** in Port **getSiteList_PORT**	

source

```
<binding name="getSiteList_SOAP_BINDING" type="y:getSiteList_PORT_TYPE">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetSiteList">
    <soap:operation soapAction="urn:#GetSiteList"/>
    <input>
      <soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      <soap:body parts="ownerId" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </input>
    <soap:operation soapAction="urn:#GetSiteList"/>
    <output>
      <soap:header message="y:headerBlock" part="header" use="encoded"
```

```
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      <soap:body parts="siteList" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
     </output>
     <fault name="getSiteListFault">
      <soap:header message="y:headerBlock" part="header" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      <soap:fault name="getSiteListFault" use="literal" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
     </fault>
    </operation>
   </binding>
```

binding **find_SOAP_BINDING**

type	**y:find_PORT_TYPE**
extensibility	<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
operations	**Find**
	extensibility **<soap:operation soapAction="urn:#Find"/>**
	input **<soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/><soap:body parts="agentPoperty" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>**
	output **<soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/><soap:body parts="agentList" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>**
used by	Service **find** in Port **find_PORT**

source

```
   <binding name="find_SOAP_BINDING" type="y:find_PORT_TYPE">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="Find">
     <soap:operation soapAction="urn:#Find"/>
     <input>
      <soap:header message="y:headerBlock" part="header" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      <soap:body parts="agentPoperty" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
     </input>
     <soap:operation soapAction="urn:#Find"/>
     <output>
      <soap:header message="y:headerBlock" part="header" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      <soap:body parts="agentList" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
     </output>
     <fault name="findFault">
      <soap:header message="y:headerBlock" part="header" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      <soap:fault name="findFault" use="literal" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
     </fault>
    </operation>
```

```
</binding>
```

porttype **register_PORT_TYPE**

operations	**Register**
	input **y:register_REQUEST**
	output **y:register_RESPONSE**
	fault **y:registration_FAULT**
used by	binding **register_SOAP_BINDING**

source

```
<portType name="register_PORT_TYPE">
  <operation name="Register">
    <input message="y:register_REQUEST"/>
    <output message="y:register_RESPONSE"/>
    <fault name="registrationFault" message="y:registration_FAULT"/>
  </operation>
</portType>
```

porttype **unregister_PORT_TYPE**

operations	**Unregister**
	fault **y:unregistration_FAULT**
	input **y:unregister_REQUEST**
used by	binding **unregister_SOAP_BINDING**

source

```
<portType name="unregister_PORT_TYPE">
  <operation name="Unregister">
    <fault name="unregistrationFault" message="y:unregistration_FAULT"/>
    <input message="y:unregister_REQUEST"/>
  </operation>
</portType>
```

porttype **updateReqistration_PORT_TYPE**

operations	**UpdateRegistration**
	input **y:updateRegistration_REQUEST**
	fault **y:updateRegistration_FAULT**
used by	binding **updateRegistration_SOAP_BINDING**

source

```
<portType name="updateReqistration_PORT_TYPE">
```

```
    <operation name="UpdateRegistration">
      <input message="y:updateRegistration_REQUEST"/>
      <fault name="updateRegistrationFault" message="y:updateRegistration_FAULT"/>
    </operation>
  </portType>
```

porttype **getSiteList_PORT_TYPE**

operations	**GetSiteList**	
	input	**y:getSiteList_REQUEST**
	output	**y:getSiteList_RESPONSE**
	fault	**y:getSiteList_FAULT**
used by	binding **getSiteList_SOAP_BINDING**	

source

```
<portType name="getSiteList_PORT_TYPE">
  <operation name="GetSiteList">
    <input message="y:getSiteList_REQUEST"/>
    <output message="y:getSiteList_RESPONSE"/>
    <fault name="getSiteListFault" message="y:getSiteList_FAULT"/>
  </operation>
</portType>
```

porttype **find_PORT_TYPE**

operations	**Find**	
	input	**y:find_REQUEST**
	output	**y:find_RESPONSE**
	fault	**y:find_FAULT**
used by	binding **find_SOAP_BINDING**	

source

```
<portType name="find_PORT_TYPE">
  <operation name="Find">
    <input message="y:find_REQUEST"/>
    <output message="y:find_RESPONSE"/>
    <fault name="findFault" message="y:find_FAULT"/>
  </operation>
</portType>
```

message **headerBlock**

parts **header**

type **ns:MandatoryHeaderBlockType**

source
```
<message name="headerBlock">
  <part name="header" type="ns:MandatoryHeaderBlockType"/>
</message>
```

message **register_REQUEST**

parts **agentInfo**

type **ns:AgentInfo**

used by PortType **register_PORT_TYPE** in Operation **Register**

source
```
<message name="register_REQUEST">
  <part name="agentInfo" type="ns:AgentInfo"/>
</message>
```

message **register_RESPONSE**

parts **agentId**

type **ns:ComponentId**

used by PortType **register_PORT_TYPE** in Operation **Register**

source
```
<message name="register_RESPONSE">
  <part name="agentId" type="ns:ComponentId"/>
</message>
```

message **registration_FAULT**

parts **registrationFault**

type **ns:RegistrationFault**

used by PortType **register_PORT_TYPE** in Operation **Register**

source
```
<message name="registration_FAULT">
  <part name="registrationFault" type="ns:RegistrationFault"/>
</message>
```

message **unregister_REQUEST**

parts

used by PortType **unregister_PORT_TYPE** in Operation **Unregister**

source
```
<message name="unregister_REQUEST"/>
```

message **unregistration_FAULT**

parts	**unregistrationFault**	
	type	**ns:ServiceFault**
used by	PortType **unregister_PORT_TYPE** in Operation **Unregister**	

source

```
<message name="unregistration_FAULT">
  <part name="unregistrationFault" type="ns:ServiceFault"/>
</message>
```

message **updateRegistration_REQUEST**

parts	**agentInfo**	
	type	**ns:AgentInfo**
used by	PortType **updateRegistration_PORT_TYPE** in Operation **UpdateRegistration**	

source

```
<message name="updateRegistration_REQUEST">
  <part name="agentInfo" type="ns:AgentInfo"/>
</message>
```

message **updateRegistration_FAULT**

parts	**updateRegistrationFault**	
	type	**ns:ServiceFault**
used by	PortType **updateRegistration_PORT_TYPE** in Operation **UpdateRegistration**	

source

```
<message name="updateRegistration_FAULT">
  <part name="updateRegistrationFault" type="ns:ServiceFault"/>
</message>
```

message **getSiteList_REQUEST**

parts	**ownerId**	
	type	**xs:string**
used by	PortType **getSiteList_PORT_TYPE** in Operation **GetSiteList**	

source

```
<message name="getSiteList_REQUEST">
  <part name="ownerId" type="xs:string"/>
</message>
```

message **getSiteList_RESPONSE**

parts	**siteList**
	type **ns:SiteList**
used by	PortType **getSiteList_PORT_TYPE** in Operation **GetSiteList**
source	

```
<message name="getSiteList_RESPONSE">
  <part name="siteList" type="ns:SiteList"/>
</message>
```

message **getSiteList_FAULT**

parts	**getSiteListFault**
	type **ns:ServiceFault**
used by	PortType **getSiteList_PORT_TYPE** in Operation **GetSiteList**
source	

```
<message name="getSiteList_FAULT">
  <part name="getSiteListFault" type="ns:ServiceFault"/>
</message>
```

message **find_REQUEST**

parts	**agentProperty**
	type **ns:AgentProperty**
used by	PortType **find_PORT_TYPE** in Operation **Find**
source	

```
<message name="find_REQUEST">
  <part name="agentProperty" type="ns:AgentProperty"/>
</message>
```

message **find_RESPONSE**

parts	**agentList**
	type **ns:AgentList**
used by	PortType **find_PORT_TYPE** in Operation **Find**
source	

```
<message name="find_RESPONSE">
  <part name="agentList" type="ns:AgentList"/>
</message>
```

message **find_FAULT**

parts	**findFault**
	type **ns:DiscoveryFault**

used by	PortType **find_PORT_TYPE** in Operation **Find**

source	`<message name="find_FAULT">`
	`  <part name="findFault" type="ns:DiscoveryFault"/>`
	`</message>`

9.2 Delegate Web Services Definition

WSDL **Delegate.wsdl**

WSDL location:	**.\XSD\Delegate.wsdl**
targetnamespace:	**http://supervisor.namespace**

services	bindings	porttypes	messages
getAgentInfo	**getAgentInfo_SOAP_BINDING**	**getAgentInfo_PORT_TYPE**	**getAgentInfo_FAULT**
perform	**perform_SOAP_BINDING**	**perform_PORT_TYPE**	**getAgentInfo_REQUEST**
query	**query_SOAP_BINDING**	**query_PORT_TYPE**	**getAgentInfo_RESPONSE**
subscribe	**subscribe_SOAP_BINDING**	**subscribe_PORT_TYPE**	**headerBlock**
unsubscribe	**unsubscribe_SOAP_BINDING**	**unsubscribe_PORT_TYPE**	**perform_FAULT**
			perform_REQUEST
			perform_RESPONSE
			query_FAULT
			query_REQUEST
			query_RESPONSE
			subscribe_FAULT
			subscribe_REQUEST
			unsubscribe_FAULT
			unsubscribe_REQUEST

service **getAgentInfo**

ports	**getAgentInfo_PORT**	
	binding	**y:getAgentInfo_SOAP_BINDING**
	extensibility	**`<soap:address location="No Target Adress"/>`**
source	`<service name="getAgentInfo">`	

```
  <port name="getAgentInfo_PORT" binding="y:getAgentInfo_SOAP_BINDING">
    <soap:address location="No Target Adress"/>
  </port>
</service>
```

service **query**

ports	**query_PORT**	
	binding	**y:query_SOAP_BINDING**
	extensibility	**<soap:address location="No Target Adress"/>**

source

```
<service name="query">
  <port name="query_PORT" binding="y:query_SOAP_BINDING">
    <soap:address location="No Target Adress"/>
  </port>
</service>
```

service **subscribe**

ports	**subscribe_PORT**	
	binding	**y:subscribe_SOAP_BINDING**
	extensibility	**<soap:address location="No Target Adress"/>**

source

```
<service name="subscribe">
  <port name="subscribe_PORT" binding="y:subscribe_SOAP_BINDING">
    <soap:address location="No Target Adress"/>
  </port>
</service>
```

service **unsubscribe**

ports	**unsubscribe_PORT**	
	binding	**y:unsubscribe_SOAP_BINDING**
	extensibility	**<soap:address location="No Target Adress"/>**

source

```
<service name="unsubscribe">
  <port name="unsubscribe_PORT" binding="y:unsubscribe_SOAP_BINDING">
    <soap:address location="No Target Adress"/>
  </port>
</service>
```

service **perform**

ports	**perform_PORT**	
	binding	**y:perform_SOAP_BINDING**
	extensibility	**<soap:address location="No Target Adress"/>**

source

```
<service name="perform">
  <port name="perform_PORT" binding="y:perform_SOAP_BINDING">
    <soap:address location="No Target Adress"/>
  </port>
</service>
```

binding **getAgentInfo_SOAP_BINDING**

type	**y:getAgentInfo_PORT_TYPE**	
extensibility	<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>	
operations	**GetAgentInfo**	
	extensibility	
	input	**<soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>**
	output	**<soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/><soap:body parts="agentInfo" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>**
used by	Service **getAgentInfo** in Port **getAgentInfo_PORT**	

source

```
<binding name="getAgentInfo_SOAP_BINDING" type="y:getAgentInfo_PORT_TYPE">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetAgentInfo">
    <input>
      <soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </input>
    <output>
      <soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      <soap:body parts="agentInfo" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </output>
    <fault name="getAgentInfoFault">
      <soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      <soap:fault name="getAgentInfoFault" use="literal" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </fault>
  </operation>
</binding>
```

binding **query_SOAP_BINDING**

type	**y:query_PORT_TYPE**
extensibility	<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
operations	**Query** **extensibility** **input** **<soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/><soap:body parts="supRequest" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>** **output** **<soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/><soap:body parts="supResponse" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>**
used by	Service **query** in Port **query_PORT**
source	

```
<binding name="query_SOAP_BINDING" type="y:query_PORT_TYPE">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="Query">
    <input>
      <soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      <soap:body parts="supRequest" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </input>
    <output>
      <soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      <soap:body parts="supResponse" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </output>
    <fault name="queryFault">
      <soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      <soap:fault name="queryFault" use="literal" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </fault>
  </operation>
</binding>
```

binding **subscribe_SOAP_BINDING**

type	**y:subscribe_PORT_TYPE**
extensibility	<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
operations	**Subscribe** **extensibility** **input** **<soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/><soap:body parts="supRequest" use="encoded"**

encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>

out put

used by Service **subscribe** in Port **subscribe_PORT**

source

```
<binding name="subscribe_SOAP_BINDING" type="y:subscribe_PORT_TYPE">
 <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="Subscribe">
  <input>
   <soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   <soap:body parts="supRequest" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <fault name="subscribeFault">
   <soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   <soap:fault name="subscribeFault" use="literal" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </fault>
 </operation>
</binding>
```

binding **unsubscribe_SOAP_BINDING**

type **y:unsubscribe_PORT_TYPE**

extensibility `<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>`

operations **Unsubscribe**

extensibility

input **<soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/><soap:body parts="supRequest" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>**

out put

used by Service **unsubscribe** in Port **unsubscribe_PORT**

source

```
<binding name="unsubscribe_SOAP_BINDING" type="y:unsubscribe_PORT_TYPE">
 <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="Unsubscribe">
  <input>
   <soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   <soap:body parts="supRequest" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <fault name="unsubscribeFault">
   <soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   <soap:fault name="unsubscribeFault" use="literal" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </fault>
  </operation>
</binding>
```

binding **perform_SOAP_BINDING**

type	y:perform_PORT_TYPE
extensibility	<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
operations	**Perform**
	extensibility
	input <soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/><soap:body parts="supRequest" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
	output <soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/><soap:body parts="supResponse" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
used by	Service perform in Port perform_PORT

source

```
<binding name="perform_SOAP_BINDING" type="y:perform_PORT_TYPE">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="Perform">
    <input>
      <soap:header message="y:headerBlock" part="header" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      <soap:body parts="supRequest" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </input>
    <output>
      <soap:header message="y:headerBlock" part="header" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      <soap:body parts="supResponse" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </output>
    <fault name="performFault">
      <soap:header message="y:headerBlock" part="header" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      <soap:fault name="performFault" use="literal" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </fault>
  </operation>
</binding>
```

porttype **getAgentInfo_PORT_TYPE**

operations	**GetAgentInfo**
	input y:getAgentInfo_REQUEST
	out y:getAgentInfo_RESPONSE

	put	
	fault	**y:getAgentInfo_FAULT**
used by	binding **getAgentInfo_SOAP_BINDING**	

source

```
<portType name="getAgentInfo_PORT_TYPE">
  <operation name="GetAgentInfo">
    <input message="y:getAgentInfo_REQUEST"/>
    <output message="y:getAgentInfo_RESPONSE"/>
    <fault name="getAgentInfoFault" message="y:getAgentInfo_FAULT"/>
  </operation>
</portType>
```

porttype **query_PORT_TYPE**

operations	**Query**	
	input	**y:query_REQUEST**
	output	**y:query_RESPONSE**
	fault	**y:query_FAULT**
used by	binding **query_SOAP_BINDING**	

source

```
<portType name="query_PORT_TYPE">
  <operation name="Query">
    <input message="y:query_REQUEST"/>
    <output message="y:query_RESPONSE"/>
    <fault name="queryFault" message="y:query_FAULT"/>
  </operation>
</portType>
```

porttype **subscribe_PORT_TYPE**

operations	**Subscribe**	
	input	**y:subscribe_REQUEST**
	fault	**y:subscribe_FAULT**
used by	binding **subscribe_SOAP_BINDING**	

source

```
<portType name="subscribe_PORT_TYPE">
  <operation name="Subscribe">
    <input message="y:subscribe_REQUEST"/>
    <fault name="subscribeFault" message="y:subscribe_FAULT"/>
  </operation>
</portType>
```

porttype **unsubscribe_PORT_TYPE**

operations	**Unsubscribe**	
	input	**y:unsubscribe_REQUEST**
	fault	**y:unsubscribe_FAULT**
used by	binding **unsubscribe_SOAP_BINDING**	

source

```
<portType name="unsubscribe_PORT_TYPE">
  <operation name="Unsubscribe">
    <input message="y:unsubscribe_REQUEST"/>
    <fault name="unsubscribeFault" message="y:unsubscribe_FAULT"/>
  </operation>
</portType>
```

porttype **perform_PORT_TYPE**

operations	**Perform**	
	input	**y:perform_REQUEST**
	output	**y:perform_RESPONSE**
	fault	**y:perform_FAULT**
used by	binding **perform_SOAP_BINDING**	

source

```
<portType name="perform_PORT_TYPE">
  <operation name="Perform">
    <input message="y:perform_REQUEST"/>
    <output message="y:perform_RESPONSE"/>
    <fault name="performFault" message="y:perform_FAULT"/>
  </operation>
</portType>
```

message **headerBlock**

parts	**header**	
	type	**ns:MandatoryHeaderBlockType**

source

```
<message name="headerBlock">
  <part name="header" type="ns:MandatoryHeaderBlockType"/>
</message>
```

message **getAgentInfo_REQUEST**

parts

used by PortType **getAgentInfo_PORT_TYPE** in Operation **GetAgentInfo**

source
```
<message name="getAgentInfo_REQUEST"/>
```

message **getAgentInfo_RESPONSE**

parts **agentInfo**

type **ns:AgentInfo**

used by PortType **getAgentInfo_PORT_TYPE** in Operation **GetAgentInfo**

source
```
<message name="getAgentInfo_RESPONSE">
  <part name="agentInfo" type="ns:AgentInfo"/>
</message>
```

message **getAgentInfo_FAULT**

parts **getAgentInfoFault**

type **ns:ServiceFault**

used by PortType **getAgentInfo_PORT_TYPE** in Operation **GetAgentInfo**

source
```
<message name="getAgentInfo_FAULT">
  <part name="getAgentInfoFault" type="ns:ServiceFault"/>
</message>
```

message **query_REQUEST**

parts **supRequest**

type **ns:MultipleRequest**

used by PortType **query_PORT_TYPE** in Operation **Query**

source
```
<message name="query_REQUEST">
  <part name="supRequest" type="ns:MultipleRequest"/>
</message>
```

message **query_RESPONSE**

parts **supResponse**

type **ns:MultipleResponse**

used by PortType **query_PORT_TYPE** in Operation **Query**

source
```
<message name="query_RESPONSE">
  <part name="supResponse" type="ns:MultipleResponse"/>
```

```
</message>
```

message **query_FAULT**

parts	**queryFault**	
	type	**ns:ServiceFault**
used by	PortType **query_PORT_TYPE** in Operation **Query**	

source

```
<message name="query_FAULT">
  <part name="queryFault" type="ns:ServiceFault"/>
</message>
```

message **subscribe_REQUEST**

parts	**supRequest**	
	type	**ns:MultipleRequest**
used by	PortType **subscribe_PORT_TYPE** in Operation **Subscribe**	

source

```
<message name="subscribe_REQUEST">
  <part name="supRequest" type="ns:MultipleRequest"/>
</message>
```

message **subscribe_FAULT**

parts	**subscribeFault**	
	type	**ns:ServiceFault**
used by	PortType **subscribe_PORT_TYPE** in Operation **Subscribe**	

source

```
<message name="subscribe_FAULT">
  <part name="subscribeFault" type="ns:ServiceFault"/>
</message>
```

message **unsubscribe_REQUEST**

parts	**supRequest**	
	type	**ns:MultipleRequest**
used by	PortType **unsubscribe_PORT_TYPE** in Operation **Unsubscribe**	

source

```
<message name="unsubscribe_REQUEST">
  <part name="supRequest" type="ns:MultipleRequest"/>
</message>
```

message **unsubscribe_FAULT**

parts	**unsubscribeFault**	
	type	**ns:ServiceFault**
used by	PortType **unsubscribe_PORT_TYPE** in Operation **Unsubscribe**	

source

```
<message name="unsubscribe_FAULT">
  <part name="unsubscribeFault" type="ns:ServiceFault"/>
</message>
```

message **perform_REQUEST**

parts	**supRequest**	
	type	**ns:MultipleRequest**
used by	PortType **perform_PORT_TYPE** in Operation **Perform**	

source

```
<message name="perform_REQUEST">
  <part name="supRequest" type="ns:MultipleRequest"/>
</message>
```

message **perform_RESPONSE**

parts	**supResponse**	
	type	**ns:MultipleResponse**
used by	PortType **perform_PORT_TYPE** in Operation **Perform**	

source

```
<message name="perform_RESPONSE">
  <part name="supResponse" type="ns:MultipleResponse"/>
</message>
```

message **perform_FAULT**

parts	**performFault**	
	type	**ns:ServiceFault**
used by	PortType **perform_PORT_TYPE** in Operation **Perform**	

source

```
<message name="perform_FAULT">
  <part name="performFault" type="ns:ServiceFault"/>
</message>
```

9.3 SUPERVISOR WEB SERVICES DEFINITION

WSDL **Supervisor.wsdl**

WSDL location:	**.\XSD\Supervisor.wsdl**
targetnamespace:	**http://supervisor.namespace**

services	bindings	porttypes	messages
acceptMonitoringMessage	**accept_SOAP_BINDING**	**accept_PORT_TYPE**	**accept_FAULT**
getAgentInfo	**getAgentInfo_SOAP_BINDING**	**getAgentInfo_PORT_TYPE**	**accept_REQUEST**
			getAgentInfo_FAULT
			getAgentInfo_REQUEST
			getAgentInfo_RESPONSE
			headerBlock

service **getAgentInfo**

ports **getAgentInfo_PORT**

binding	**y:getAgentInfo_SOAP_BINDING**
extensibility	**<soap:address location="No Target Adress"/>**

source

```
<service name="getAgentInfo">
  <port name="getAgentInfo_PORT" binding="y:getAgentInfo_SOAP_BINDING">
    <soap:address location="No Target Adress"/>
  </port>
</service>
```

service **acceptMonitoringMessage**

ports **accept_PORT**

binding	**y:accept_SOAP_BINDING**
extensibility	**<soap:address location="No Target Adress"/>**

source

```
<service name="acceptMonitoringMessage">
  <port name="accept_PORT" binding="y:accept_SOAP_BINDING">
    <soap:address location="No Target Adress"/>
  </port>
</service>
```

binding **getAgentInfo_SOAP_BINDING**

type	[y:getAgentInfo_PORT_TYPE]
extensibility	<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
operations	**GetAgentInfo**
	extensibility
	input **<soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>**
	output **<soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/><soap:body parts="agentInfo" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>**
used by	Service **getAgentInfo** in Port **getAgentInfo_PORT**

source

```
<binding name="getAgentInfo_SOAP_BINDING" type="y:getAgentInfo_PORT_TYPE">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetAgentInfo">
    <input>
      <soap:header message="y:headerBlock" part="header" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </input>
    <output>
      <soap:header message="y:headerBlock" part="header" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      <soap:body parts="agentInfo" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </output>
    <fault name="getAgentInfoFault">
      <soap:header message="y:headerBlock" part="header" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      <soap:fault name="getAgentInfoFault" use="literal"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </fault>
  </operation>
</binding>
```

binding **accept_SOAP_BINDING**

type	[y:accept_PORT_TYPE]
extensibility	<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
operations	**Accept**
	extensibility
	input **<soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/><soap:body parts="supResponse" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>**
	out

	put
used by	Service **acceptMonitoringMessage** in Port **accept_PORT**

source

```
<binding name="accept_SOAP_BINDING" type="y:accept_PORT_TYPE">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="Accept">
    <input>
      <soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      <soap:body parts="supResponse" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </input>
    <fault name="acceptFault">
      <soap:header message="y:headerBlock" part="header" use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      <soap:fault name="acceptFault" use="literal" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </fault>
  </operation>
</binding>
```

porttype **getAgentInfo_PORT_TYPE**

operations	**GetAgentInfo**	
	input	**y:getAgentInfo_REQUEST**
	output	**y:getAgentInfo_RESPONSE**
	fault	**y:getAgentInfo_FAULT**
used by	binding **getAgentInfo_SOAP_BINDING**	

source

```
<portType name="getAgentInfo_PORT_TYPE">
  <operation name="GetAgentInfo">
    <input message="y:getAgentInfo_REQUEST"/>
    <output message="y:getAgentInfo_RESPONSE"/>
    <fault name="getAgentInfoFault" message="y:getAgentInfo_FAULT"/>
  </operation>
</portType>
```

porttype **accept_PORT_TYPE**

operations	**Accept**	
	input	**y:accept_REQUEST**
	fault	**y:accept_FAULT**
used by	binding **accept_SOAP_BINDING**	

source

```
<portType name="accept_PORT_TYPE">
```

```
  <operation name="Accept">
    <input message="y:accept_REQUEST"/>
    <fault name="acceptFault" message="y:accept_FAULT"/>
  </operation>
</portType>
```

message **headerBlock**

parts	**header**	
	type	**ns:MandatoryHeaderBlockType**

source

```
<message name="headerBlock">
  <part name="header" type="ns:MandatoryHeaderBlockType"/>
</message>
```

message **getAgentInfo_REQUEST**

parts

used by PortType **getAgentInfo_PORT_TYPE** in Operation **GetAgentInfo**

source

```
<message name="getAgentInfo_REQUEST"/>
```

message **getAgentInfo_RESPONSE**

parts	**agentInfo**	
	type	**ns:AgentInfo**

used by PortType **getAgentInfo_PORT_TYPE** in Operation **GetAgentInfo**

source

```
<message name="getAgentInfo_RESPONSE">
  <part name="agentInfo" type="ns:AgentInfo"/>
</message>
```

message **getAgentInfo_FAULT**

parts	**getAgentInfoFault**	
	type	**ns:ServiceFault**

used by PortType **getAgentInfo_PORT_TYPE** in Operation **GetAgentInfo**

source

```
<message name="getAgentInfo_FAULT">
  <part name="getAgentInfoFault" type="ns:ServiceFault"/>
</message>
```

message **accept_REQUEST**

parts	**supResponse**	
	type	**ns:MultipleResponse**
used by	PortType **accept_PORT_TYPE** in Operation **Accept**	

source

```
<message name="accept_REQUEST">
  <part name="supResponse" type="ns:MultipleResponse"/>
</message>
```

message **accept_FAULT**

parts	**acceptFault**	
	type	**ns:ServiceFault**
used by	PortType **accept_PORT_TYPE** in Operation **Accept**	

source

```
<message name="accept_FAULT">
  <part name="acceptFault" type="ns:ServiceFault"/>
</message>
```

9.4 Information Types – XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 2 U (http://www.xmlspy.com) by toto (toto) -->
<xs:schema     targetNamespace="http://sup.types.namespace"     xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:fault="http://fault.types.namespace"    xmlns:sup="http://sup.types.namespace"    elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:import namespace="http://fault.types.namespace" schemaLocation="D:\andrey\thesis\XSD\faultTypes.xsd"/>
    <xs:simpleType name="SupervisionMessageType">
        <xs:annotation>
            <xs:documentation>Indicates Supervision Message</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="SupervisionMessage"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="ServiceMessageType">
        <xs:annotation>
            <xs:documentation>Indicates Service Message</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="ServiceMessage"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="SupervisorAgentType">
        <xs:annotation>
            <xs:documentation>Supervisor Type</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="Supervisor"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="DelegateAgentType">
        <xs:annotation>
```

```
            <xs:documentation>Delegate Type</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="Delegate"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="QueryResponseInteractionModel">
        <xs:annotation>
            <xs:documentation>Query/Response Interaction Model</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="QueryResponse"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="EventSubscribeInteractionModel">
        <xs:annotation>
            <xs:documentation>Subscribe for Event Interaction Model</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="EventSubscribe"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="PerformInteractionModel">
        <xs:annotation>
            <xs:documentation>Perform Command Interaction Model</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="Perform"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:complexType name="ComponentId">
        <xs:annotation>
            <xs:documentation>Component Location Identification</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="coreId" type="xs:string"/>
            <xs:element name="ownerId" type="xs:string"/>
            <xs:element name="siteId" type="xs:string"/>
            <xs:element name="agentId" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="AgentType">
        <xs:annotation>
            <xs:documentation>Type of Agent</xs:documentation>
        </xs:annotation>
        <xs:choice>
            <xs:choice>
                <xs:element name="stype" type="sup:SupervisorAgentType"/>
                <xs:element name="dtype" type="sup:DelegateAgentType"/>
            </xs:choice>
            <xs:sequence>
                <xs:element name="stype" type="sup:SupervisorAgentType"/>
                <xs:element name="dtype" type="sup:DelegateAgentType"/>
            </xs:sequence>
        </xs:choice>
    </xs:complexType>
    <xs:complexType name="MessageType">
        <xs:annotation>
            <xs:documentation>Type of Message Service|Supervision</xs:documentation>
        </xs:annotation>
        <xs:choice>
            <xs:element name="sertype" type="sup:ServiceMessageType"/>
            <xs:element name="suptype" type="sup:SupervisionMessageType"/>
        </xs:choice>
    </xs:complexType>
    <xs:complexType name="InteractionModel">
        <xs:annotation>
            <xs:documentation>Interaction Model</xs:documentation>
        </xs:annotation>
        <xs:choice>
            <xs:sequence>
            <xs:element name="qrtype" type="sup:QueryResponseInteractionModel"/>
            <xs:choice>
                <xs:element name="estype" type="sup:EventSubscribeInteractionModel" minOccurs="0"/>
                <xs:element name="ptype" type="sup:PerformInteractionModel" minOccurs="0"/>
            </xs:choice>
        </xs:sequence>
        <xs:sequence>
            <xs:element name="estype" type="sup:EventSubscribeInteractionModel"/>
            <xs:choice>
                <xs:element name="qrtype" type="sup:QueryResponseInteractionModel" minOccurs="0"/>
                <xs:element name="ptype" type="sup:PerformInteractionModel" minOccurs="0"/>
            </xs:choice>
        </xs:sequence>
        <xs:sequence>
            <xs:element name="ptype" type="sup:PerformInteractionModel"/>
            <xs:sequence>
                <xs:element name="qrtype" type="sup:QueryResponseInteractionModel" minOccurs="0"/>
                <xs:element name="estype" type="sup:EventSubscribeInteractionModel" minOccurs="0"/>
            </xs:sequence>
        </xs:sequence>
    </xs:choice>
</xs:complexType>
<xs:complexType name="SupOperationDefinition">
    <xs:annotation>
        <xs:documentation>Definition of the Supervision Operations</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="namespace" type="xs:string"/>
        <xs:element name="schemaLocation" type="xs:anyURI"/>
        <xs:element name="input" type="sup:MultipleRequest"/>
        <xs:element name="output" type="sup:MultipleResponse"/>
        <xs:element name="fault" type="fault:SupervisionFault" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="interactionModel" type="sup:InteractionModel"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="AgentInfo">
    <xs:annotation>
        <xs:documentation>Agent Information</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="description" type="xs:string"/>
        <xs:element name="componentId" type="sup:ComponentId"/>
        <xs:element name="url" type="xs:anyURI"/>
        <xs:element name="availability" type="xs:boolean"/>
        <xs:element name="agentType" type="sup:AgentType"/>
        <xs:element name="operations">
            <xs:complexType>
                <xs:choice maxOccurs="unbounded">
                    <xs:sequence>
                        <xs:element name="opDefinitionLocation" type="xs:anyURI"/>
                    </xs:sequence>
                    <xs:sequence>
                        <xs:element name="opDefinition" type="sup:SupOperationDefinition"/>
                    </xs:sequence>
                </xs:choice>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="MandatoryHeaderBlockType">
    <xs:annotation>
        <xs:documentation>Structure of Mandatory Header Block</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="sourceComponentId" type="sup:ComponentId"/>
        <xs:element name="destinationComponentId" type="sup:ComponentId"/>
        <xs:element name="timestamp" type="xs:long"/>
        <xs:element name="category" type="sup:MessageType"/>
    </xs:sequence>
```

```
</xs:complexType>
<xs:complexType name="AgentProperty">
    <xs:annotation>
        <xs:documentation>Agent Property</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="name" type="xs:string" minOccurs="0"/>
        <xs:element name="description" type="xs:string" minOccurs="0"/>
        <xs:element name="componentId" minOccurs="0">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="coreId" type="xs:string" minOccurs="0"/>
                    <xs:element name="ownerId" type="xs:string" minOccurs="0"/>
                    <xs:element name="siteId" type="xs:string" minOccurs="0"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="url" type="xs:anyURI" minOccurs="0"/>
        <xs:element name="availability" type="xs:boolean" minOccurs="0"/>
        <xs:element name="agentType" type="sup:AgentType" minOccurs="0"/>
        <xs:element name="operations" minOccurs="0">
            <xs:complexType>
                <xs:choice maxOccurs="unbounded">
                    <xs:sequence>
                        <xs:element name="opDefinitionLocation" type="xs:anyURI"/>
                    </xs:sequence>
                    <xs:sequence>
                        <xs:element name="opDefinition" type="sup:SupOperationDefinition"/>
                    </xs:sequence>
                </xs:choice>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="AgentList">
    <xs:annotation>
        <xs:documentation>List of Agent Information</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="agentInfo" type="sup:AgentInfo" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="SiteList">
    <xs:annotation>
        <xs:documentation>List of Sites for Given Owner Id</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="ownerId" type="xs:string"/>
        <xs:sequence>
            <xs:element name="siteId" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="SimpleParameter">
    <xs:annotation>
        <xs:documentation>Geneneral Form of Supervision Parameter</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="type" type="xs:string"/>
        <xs:element name="value" type="xs:anyType"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="NestedParameter">
    <xs:annotation>
        <xs:documentation>Structure of Nested Parameter in General Form</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="parameter" type="sup:SimpleParameter" maxOccurs="unbounded"/>
        <xs:element name="nest" type="sup:NestedParameter" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
```

```
    <xs:complexType name="SingleResponse">
        <xs:annotation>
            <xs:documentation>Geneneral Form of Supervision Response for Single Enty</xs:documentation>
        </xs:annotation>
        <xs:complexContent>
            <xs:extension base="sup:NestedParameter"/>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="MultipleResponse">
        <xs:annotation>
            <xs:documentation>Geneneral Form of Supervision Response for Multiple Entries</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="response" type="sup:SingleResponse" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="SingleRequest">
        <xs:annotation>
            <xs:documentation>General Form of Single Supervision Operation Request</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="requestParameter" type="sup:NestedParameter" minOccurs="0"/>
            <xs:element name="requestParametreType" type="xs:anyType"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="MultipleRequest">
        <xs:annotation>
            <xs:documentation>General Form of Multiple Supervision Operations Request</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="request" type="sup:SingleRequest" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="agentInformation" type="sup:AgentInfo">
        <xs:annotation>
            <xs:documentation>Instance of Agent Information</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:element name="mandatoryHeaderBlock" type="sup:MandatoryHeaderBlockType">
        <xs:annotation>
            <xs:documentation>Instance of Mandatory Header Block</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:element name="supervisionResponse">
        <xs:annotation>
            <xs:documentation>Instance of Supervision Response in General Form</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:complexContent>
                <xs:restriction base="sup:MultipleResponse">
                    <xs:sequence>
                        <xs:element         name="response"         type="sup:SingleResponse"         minOccurs="0"
maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:restriction>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="supervisionRequest">
        <xs:annotation>
            <xs:documentation>Instance of Request for Supervision Oprations in General Form</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:complexContent>
                <xs:restriction base="sup:MultipleRequest">
                    <xs:sequence>
                        <xs:element          name="request"          type="sup:SingleRequest"          minOccurs="0"
maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:restriction>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="supOperation">
        <xs:annotation>
            <xs:documentation>Instance of Supervision Operation Definition</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:complexContent>
                <xs:restriction base="sup:SupOperationDefinition">
                    <xs:sequence>
                        <xs:element name="name" type="xs:string"/>
                        <xs:element name="input" type="sup:MultipleRequest"/>
                        <xs:element name="output" type="sup:MultipleResponse"/>
                        <xs:element          name="faults"          type="fault:SupervisionFault"          minOccurs="0"
maxOccurs="unbounded"/>
                        <xs:element name="interactionModel" type="sup:InteractionModel"/>
                    </xs:sequence>
                </xs:restriction>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

9.5 FREE DISK SPACE CHECK EXAMPLE – XML SCHEMA

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 2 U (http://www.xmlspy.com) by toto (toto) -->
<xs:schema targetNamespace="http://disk.example.namespace" xmlns:fault="http://fault.types.namespace"
xmlns:sup="http://sup.types.namespace" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:sys="http://disk.example.namespace" elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:import namespace="http://sup.types.namespace" schemaLocation="D:\andrey\thesis\XSD\supTypes.xsd"/>
    <xs:import namespace="http://fault.types.namespace" schemaLocation="D:\andrey\thesis\XSD\faultTypes.xsd"/>
    <xs:simpleType name="HostName">
        <xs:annotation>
            <xs:documentation>List of Hosts for Disk Info Example</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="King"/>
            <xs:enumeration value="Moorcock"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="DiskName">
        <xs:annotation>
            <xs:documentation>List of DiskNames</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="C:"/>
            <xs:enumeration value="D:"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="FreeSpace">
        <xs:annotation>
            <xs:documentation>Supervised Parameter Typr to be Requested/Returned</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:unsignedInt"/>
    </xs:simpleType>
    <xs:complexType name="DiskRequest">
        <xs:annotation>
            <xs:documentation>Example of Request Definition</xs:documentation>
        </xs:annotation>
        <xs:complexContent>
            <xs:restriction base="sup:MultipleRequest">
                <xs:sequence>
                    <xs:element name="request" maxOccurs="unbounded">
                        <xs:complexType>
                            <xs:complexContent>
                                <xs:restriction base="sup:SingleRequest">
                                    <xs:sequence>
                    <xs:element name="requestParameter">
                      <xs:complexType>
                        <xs:complexContent>
                          <xs:restriction base="sup:NestedParameter">
                            <xs:sequence>
                              <xs:element name="parameter" maxOccurs="unbounded">
                                <xs:complexType>
                                  <xs:complexContent>
                                    <xs:restriction base="sup:SimpleParameter">
                                      <xs:sequence>
                                        <xs:element name="type"
type="xs:string" fixed="sys:HostName"/>
                                        <xs:element name="value"
type="sys:HostName"/>
                                      </xs:sequence>
                                    </xs:restriction>
                                  </xs:complexContent>
                                </xs:complexType>
                              </xs:element>
                              <xs:element name="nest" maxOccurs="unbounded">
                                <xs:complexType>
                                  <xs:complexContent>
                                    <xs:restriction base="sup:NestedParameter">
                                      <xs:sequence>
                                        <xs:element name="parameter"
maxOccurs="unbounded">
                                          <xs:complexType>
                                              <xs:complexContent>
                                                  <xs:restriction
base="sup:SimpleParameter">

  <xs:sequence>

  <xs:element name="type" type="xs:string" fixed="sys:DiskName"/>

  <xs:element name="value" type="sys:DiskName"/>

  </xs:sequence>
                                                  </xs:restriction>
                                              </xs:complexContent>
                                          </xs:complexType>
                                        </xs:element>
                                        <xs:element name="nest"
type="sup:NestedParameter" minOccurs="0" maxOccurs="0"/>
                                      </xs:sequence>
                                    </xs:restriction>
                                  </xs:complexContent>
                                </xs:complexType>
                              </xs:element>
                            </xs:sequence>
                          </xs:restriction>
                        </xs:complexContent>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="requestParametreType" type="xs:string"
default="sys:FreeSpace"/>
                  </xs:sequence>
                </xs:restriction>
              </xs:complexContent>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="DiskResponse">
    <xs:annotation>
      <xs:documentation>Example of Response Definition</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:restriction base="sup:MultipleResponse">
```

```
        <xs:sequence>
          <xs:element name="response" maxOccurs="unbounded">
            <xs:complexType>
              <xs:complexContent>
                <xs:restriction base="sup:SingleResponse">
                  <xs:sequence>
                    <xs:element name="parameter">
                      <xs:complexType>
                        <xs:complexContent>
                          <xs:restriction base="sup:SimpleParameter">
                            <xs:sequence>
                              <xs:element name="type" type="xs:string"
fixed="sys:HostName"/>
                              <xs:element name="value" type="sys:HostName"/>
                            </xs:sequence>
                          </xs:restriction>
                        </xs:complexContent>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="nest">
                      <xs:complexType>
                        <xs:complexContent>
                          <xs:restriction base="sup:NestedParameter">
                            <xs:sequence>
                              <xs:element name="parameter">
                                <xs:complexType>
                                  <xs:complexContent>
                                    <xs:restriction base="sup:SimpleParameter">
                                      <xs:sequence>
                                        <xs:element name="type"
type="xs:string" fixed="sys:DiskName"/>
                                        <xs:element name="value"

type="sys:DiskName"/>

                                      </xs:sequence>
                                    </xs:restriction>
                                  </xs:complexContent>
                                </xs:complexType>
                              </xs:element>
                              <xs:element name="nest">
                                <xs:complexType>
                                  <xs:complexContent>
                                    <xs:restriction base="sup:NestedParameter">
                                      <xs:sequence>
                                        <xs:element name="parameter">
                                          <xs:complexType>
                                              <xs:complexContent>
                                                  <xs:restriction
base="sup:SimpleParameter">

  <xs:sequence>

  <xs:element name="type" type="xs:string" fixed="sys:FreeSpace"/>

  <xs:element name="value" type="sys:FreeSpace"/>

  </xs:sequence>
                                                  </xs:restriction>
                                              </xs:complexContent>
                                          </xs:complexType>
                                        </xs:element>
                                        <xs:element name="nest"
type="sup:NestedParameter" minOccurs="0" maxOccurs="0"/>
                                      </xs:sequence>
                                    </xs:restriction>
                                  </xs:complexContent>
                                </xs:complexType>
                              </xs:element>
                            </xs:sequence>
                          </xs:restriction>
                        </xs:complexContent>
                      </xs:complexType>
```

```
                                        </xs:element>
                                    </xs:sequence>
                                </xs:restriction>
                            </xs:complexContent>
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
            </xs:restriction>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="DiskFault" abstract="false" block="restriction" final="restriction" mixed="false">
        <xs:annotation>
            <xs:documentation>Example of Fault Definition</xs:documentation>
        </xs:annotation>
        <xs:complexContent mixed="false">
            <xs:restriction base="fault:SupervisionFault">
                <xs:sequence>
                    <xs:element name="faultReason" type="xs:string" fixed="NetworkFalure"/>
                    <xs:element name="description" type="xs:string" fixed="Network is not Available"/>
                </xs:sequence>
            </xs:restriction>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="DiskOp">
        <xs:annotation>
            <xs:documentation>Example of Operation Definition</xs:documentation>
        </xs:annotation>
        <xs:complexContent>
            <xs:restriction base="sup:SupOperationDefinition">
                <xs:sequence>
                    <xs:element name="name" type="xs:string" fixed="DiskFreeSpaceCheck"/>
                    <xs:element name="namespace" type="xs:string" fixed='sys="http://disk.example.namespace"'/>
                    <xs:element name="schemaLocation" type="xs:anyURI"
default="D:\andrey\thesis\XSD\diskEx.xsd"/>
                    <xs:element name="input" type="sys:DiskRequest"/>
                    <xs:element name="output" type="sys:DiskResponse"/>
                    <xs:element name="fault" type="sys:DiskFault"/>
                    <xs:element name="interactionModel">
                        <xs:complexType>
                            <xs:complexContent>
                                <xs:restriction base="sup:InteractionModel">
                                    <xs:choice>
                                        <xs:sequence>
                                            <xs:element name="qrtype" type="sup:QueryResponseInteractionModel"/>
                                        </xs:sequence>
                                    </xs:choice>
                                </xs:restriction>
                            </xs:complexContent>
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
            </xs:restriction>
        </xs:complexContent>
    </xs:complexType>
    <xs:element name="diskReq" type="sys:DiskRequest">
        <xs:annotation>
            <xs:documentation>Example of Local Time Request</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:element name="diskRes" type="sys:DiskResponse">
        <xs:annotation>
            <xs:documentation>Example of Local Time Response</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:element name="opDef" type="sys:DiskOp">
        <xs:annotation>
            <xs:documentation>Example of Local Time Operation Definition</xs:documentation>
        </xs:annotation>
    </xs:element>
</xs:schema>
```