Interdisciplinary Information Sciences Vol. 17, No. 1 (2011) 19–37
©Graduate School of Information Sciences, Tohoku University
ISSN 1340-9050 print/1347-6157 online
DOI 10.4036/iis.2011.19

Design and Implementation of a Reusable Knowledge Model for Supporting the Network Management Functions

Sameera ABAR[1,*] and Tetsuo KINOSHITA[2,*]

[1]*Graduate School of Information Sciences, Tohoku University, Sendai, Japan*
[2]*Research Institute of Electrical Communication, Tohoku University, Sendai, Japan*

Received December 9, 2008; final version accepted January 5, 2011

The focus of our work is the elicitation of communication network systems' knowledge resources in a generic and reusable manner for providing the automated support to network management tasks. Key features of the proposed knowledge model are: ontological representation of static domain-content and management-expertise encoded as the core knowledge of distributed multi-agent architecture. Our emphasis has primarily been on the modularization of resource knowledge to facilitate its reuse in a flexible manner. To demonstrate the effectiveness of proposed scheme, we have implemented an experimental network in our laboratory, and the devised knowledge model has been deployed through multi-agent based middleware layer in the prototype system. A couple of application scenarios have been designed for testing with the prototype system. Experimental results confirm a marked reduction in the workloads of the network operator with our system providing the automated support to network management functions. Validation of the reusability/modifiability aspects of our system illustrates the flexible manipulation of knowledge fragments within diverse application contexts. We envisage our knowledge modeling approach as the first step towards the comprehensive knowledge acquisition, representation, and dissemination in the communication network management domain.

KEYWORDS: Knowledge acquisition/representation/reuse, domain ontology, multi-agent middleware, distributed problem-solving, network management

1. Research Overview

With the unprecedented growth in size and complexity of heterogeneous and globally distributed communication networks, the development of intelligent and adaptive tools is the indispensable requirement for automated network management functions. Typically, the network management functions translate into fault detection and diagnosis tasks as well as performance measurements across applications, servers and network devices. Managing huge distributed networks in order to ensure that the system operates within desirable parameters is an extremely cumbersome task and poses many challenges for the network administrators. Even though failures in large communication networks are unavoidable, quick detection and identification of the causes of failure can fortify these systems, making them robust with more reliable operations, thereby ultimately increasing the level of confidence in the services they provide. Hence, for the significant administrative overhead reduction and increased robustness, the errors and failures must be worked around in an autonomous fashion for restoring and optimizing the normal state of the network. There is a demand for networks whose components are smarter than they are now to allow for learning about evolving traffic patterns, error conditions, and emerging attacks. Ultimately, such networks should be able to ward-off attacks by reasoning using deep knowledge about networking, communications, and prior forms of attack. Therefore, detailed description of the network management domain resources and underlying task specific knowledge is an essential mainstream to be taken into account. The domain knowledge resources consist of objects which represent the application domain data, and the definitions of behavior that are essential for its manipulation. The challenge of knowledge acquisition is to obtain and understand the domain knowledge resources that are required to solve the tasks in the given environment.

The scope of our work has been confined to SNMP-managed (Simple Network Management Protocol) TCP/IP-based (Internet Protocol Suite: Transmission Control Protocol and Internet Protocol) globally distributed communication network systems. This domain is inherently very complex (it is a heterogeneous and incompatible multi-vendor environment) as well as knowledge-intensive (diverse range of knowledge and data are present). Typically, the tasks of an NMS (network management system) deal with collecting and analyzing information about the network system, including both hardware resources (e.g. configuration of a router) and software (e.g. number of incoming IP-packets to a router). Deeper understanding of the network state can be acquired from the wide range of network information, which must be analyzed and interpreted by the network experts and management applications. The pre-requisite for the

[*] Corresponding author. E-mail: sameera@ka.riec.tohoku.ac.jp, kino@riec.tohoku.ac.jp

automation of management functions is the detailed interpretation of network-related information and knowledge resources.

Traditionally, knowledge engineering was viewed as a process of “extracting” knowledge from a human expert and transferring it to the machine in computational form. Today, knowledge engineering is approached as a sophisticated modeling activity, and inference or reasoning algorithms are used to solve problems with the help of this knowledge. The applications are characterized by the tasks and domains involved. Knowledge modeling can, therefore, be divided into two conceptual sub-activities: (a) modeling the domain knowledge and (b) modeling the task knowledge (Abu-Hanna and Jansweijer, 1994). The “Heuristic Classification” (Clancey, 1985) method relies on the experiential knowledge of systems and their behaviors. In contrast with other descriptions of expert systems, this method specifies the knowledge needed to solve a problem (through the useful combinations of problem solving tasks and associated sub-tasks for the purpose of sharing and reuse) independent of its representation in a particular computer language. Chandrasekaran, (1992) spotlighted the knowledge modeling approach in terms of the notion of a “task structure” which recursively links a task to alternative methods and to their sub-tasks. In the proposed work, the characterization of the network knowledge model has been performed in-line with the CommonKADS (Schreiber *et al.*, 1994) — a methodology for expertise modeling embraces the application-intensive knowledge in three types as: domain, inference, and task hierarchy.

Recently, Multi-Agent System (MAS) has emerged as a flexible way to manage the resources of distributed systems. In this paper, MAS-based approach has been adopted to deploy the functionality of the intelligent or autonomous behavior of proposed network knowledge model, to aid in the reduction of network management workload. Multi-agent systems are composed of multiple interacting agents where each agent is a coarse-grained computational system in its own right, as well as independently modifiable (Hamdi, 2006). Agents, while being well-focused on their automated tasks, provide inherently distributed solutions. While the multi-agent research area is very active and it offers an appropriate tool to tackle the network-related problems, its concerns towards this domain are not yet well covered. The scant evidence of real-world deployment of agent-based systems is clearly being considered due to the same knowledge engineering bottleneck that has been the choke-point for the widespread application of expert and intelligent systems. At present, machines and software can store the information, rank it, display it, but cannot comprehend or process it. Trying to overcome this issue led to the basic notion that if we could be able to make this information inter-linked for sharing among the agents. Therefore, it is required to make a more schematic organization of information — an organization that is more suitable for machines to operate on for the provision of various automated services. The next-generation multi-agent based Knowledge Management (KM) systems will likely rely on conceptual models in the form of ontologies — the shared conceptualizations of a domain of interests, and these structure the knowledge resources in a highly expressive manner for the purpose of efficient reuse. KM facilitates the capture, deployment, access, and reuse of information typically using contemporary technology (O’Leary, 2001).

Ontologies: the keystone of new generation of multi-agent based systems pave the way to move from a document-oriented view of the KM to a content-oriented view, where knowledge items are structured, interlinked, combined and used, thereby facilitating agent interactions and communication with the sources. Ontologies have become ubiquitous in information systems (Noy and Musen, 2004). Recently, we have seen an explosion of interest in ontologies as artifacts to represent human knowledge as critical components in knowledge management, the Semantic Web, e-commerce applications, bioinformatics and several other application areas (Brewster and O’Hara, 2004). The proposed approach utilizes a concise and consistent ontology representing the static content of the communication network system domain, capable of fitting as more as possible the technical descriptions of objects (network-related terminologies). The proposed domain ontology design in the network management domain presents a hierarchical structure, which glues together classes representing network entities and association between them. Our key concept is to specialize the agent interactions with the network systems’ knowledge resources, for autonomously and flexibly managing the network devices and resources, thereby reducing the workloads of a network administrator remarkably.

1.1 Motivation

Motivation for this research originated from the need to devise a MAS-mediated and ontology-driven knowledge-based strategy in support of the automatic provision of just-in-time and just-enough, context-dependent knowledge for actively managing the data communication network systems (Abar *et al.*, 2005). So far, a little work has been done for managing the operational knowledge of the communication network systems. Hence, the proposed idea can be regarded as an initial step towards the acquisition, representation, utilization, and sharing of widely distributed network knowledge resources.

Another reason for devising our network knowledge representation scheme stems from the fact that not many knowledge modeling techniques have been developed for the diagnostic technical domains. The main draw-back of knowledge-based systems is a need for knowledge acquisition — a well-known bottleneck for many artificial intelligence applications. Building new knowledge-based systems today usually entails constructing fresh knowledge bases from scratch. It could instead be done by assembling reusable components (Neches *et al.*, 1991). Therefore, this work can serve as a test-bed to be reused for various practical diagnostic domains. The proposed approach embodies the network-related knowledge in the form of uniformly represented semantic models, thereby providing a promising mechanism to achieve reusability and maintainability.

1.2 Relevant Literatures

Several efforts have been reported in literature as far as the automation of network management functions is concerned. However, these comparable studies focus primarily on the expert systems, or refitting the agent paradigm to management solutions. Further, these works seriously overlook many important issues regarding an ample exposition of diverse range of network knowledge resources, which is highly desirable for the efficient multi-agent interactions. A lot of prior work has been done on the data structure of managed objects, and the management of the network systems is formally done through the conventional SNMP-based polling mechanism. However, so far a concrete effort for managing the knowledge resources of communication network systems has not been presented in the contemporary research works. Since the scope of our research is confined towards exploiting the knowledge resources within the network management domain, therefore, we will mention the ones that we assess as the most relevant to our network knowledge representation scheme.

In this regard, (Li and Leon, 1998) discusses that the network-related knowledge can be formalized in a similar way as the SMI-based (Structure of Management Information) general definitions for MIBs (Management Information Base) in an ASN.1 (Abstract Syntax Notation) format, for sharing among agents in the distributed networking environment. However, the practical adoption of this approach clearly marks its limitations in comparison to the proposed scheme. KACTUS project (Bernaras *et al.*, 1998) investigates the feasibility of ontological knowledge reuse in the context of complex technical diagnostic systems. In addition to this, (Konno *et al.*, 2004) presents the Active Information Resource based network management system, however, it does not take into account the knowledge representation issues in an in-depth manner. Lemos *et al.*, (1999) led to a generic knowledge acquisition mechanism based on the concept of domain and MIB-variable based causal models, for the communication network management support systems. However this work lacks many important issues about the detailed analysis of the networks' knowledge resources as well as mapping and realization of the knowledge models. We aspire to further extend this concept towards constructing a fully functional system, to provide automated management support in an efficient manner, by elaborating the underlying knowledge resources of network systems. The proposed idea can be regarded as a foremost step towards the acquisition, representation, and sharing of widely distributed network knowledge, for providing the automated management support to communication network systems.

1.3 Paper Organization

The paper is structured as follows. Section-2 presents our formalization of the communication network-related knowledge model. The implementation considerations of multi-agent middleware system and realization of proposed knowledge model with the multi-agent approach have been illustrated in Section-3. Experimental network system and the application scenarios for performing real-time tests have been described in Section-4. Evaluation of the modifiability/reusability aspects of the proposed systems' design has been presented in Section-5. Finally, the concluding remarks follow in Section-6.

2. Proposed Knowledge Model

Our knowledge modeling approach categorizes the "network systems' data" as static domain content and dynamic status information; and the "experiential management knowledge" as task hierarchy with corresponding inference support mechanism, as illustrated in Fig. 1. The proposed knowledge modeling strategy has been constructed in-line with CommonKADS (Schreiber *et al.*, 1994) (which is a comprehensive methodology for structuring the application intensive knowledge for the reuse-based expert systems) to facilitate its reusability, shareability, and maintainability. The proposed modeling approach for knowledge acquisition, representation, and sharing results in a set of concise and logically consistent knowledge components and specifications, such as domain ontology, fault-state causal reasoning models and the associated generic task-structures for handling the diagnostic sessions. Figure 2 is the pictorial representation of the proposed knowledge-based strategy for the automated support to communication network management functions. Table 1 presents our proposed framework of the communication networks' knowledge resources.

■ Domain Factual Knowledge

Static domain-specific content and its semantics are specified in a declarative form in the proposed system. It consists of network systems' actual configurations and infrastructure; for instance, nomenclature of the network objects, IP-Addresses, Internet Domain Names, routing information, port numbers, application settings etc. The main classes or general concepts contain information about various components or nodes of the network system under consideration. In the experimental network system, the domain-specific concepts and their relationships are hierarchically organized as domain knowledge ontology. Ontologies are commonly used to capture static knowledge about some domain of interest. These domain structures enable the empirical management content to navigate through them during the course of diagnostic reasoning process.

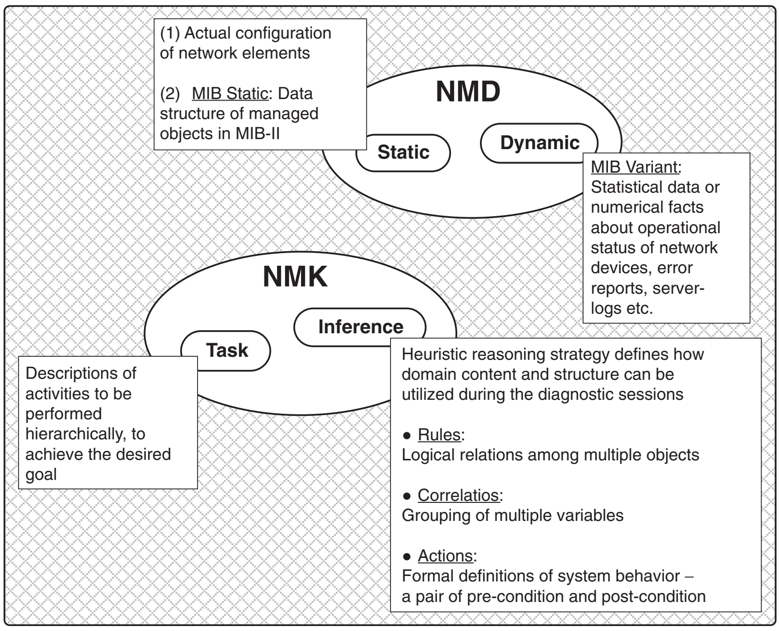
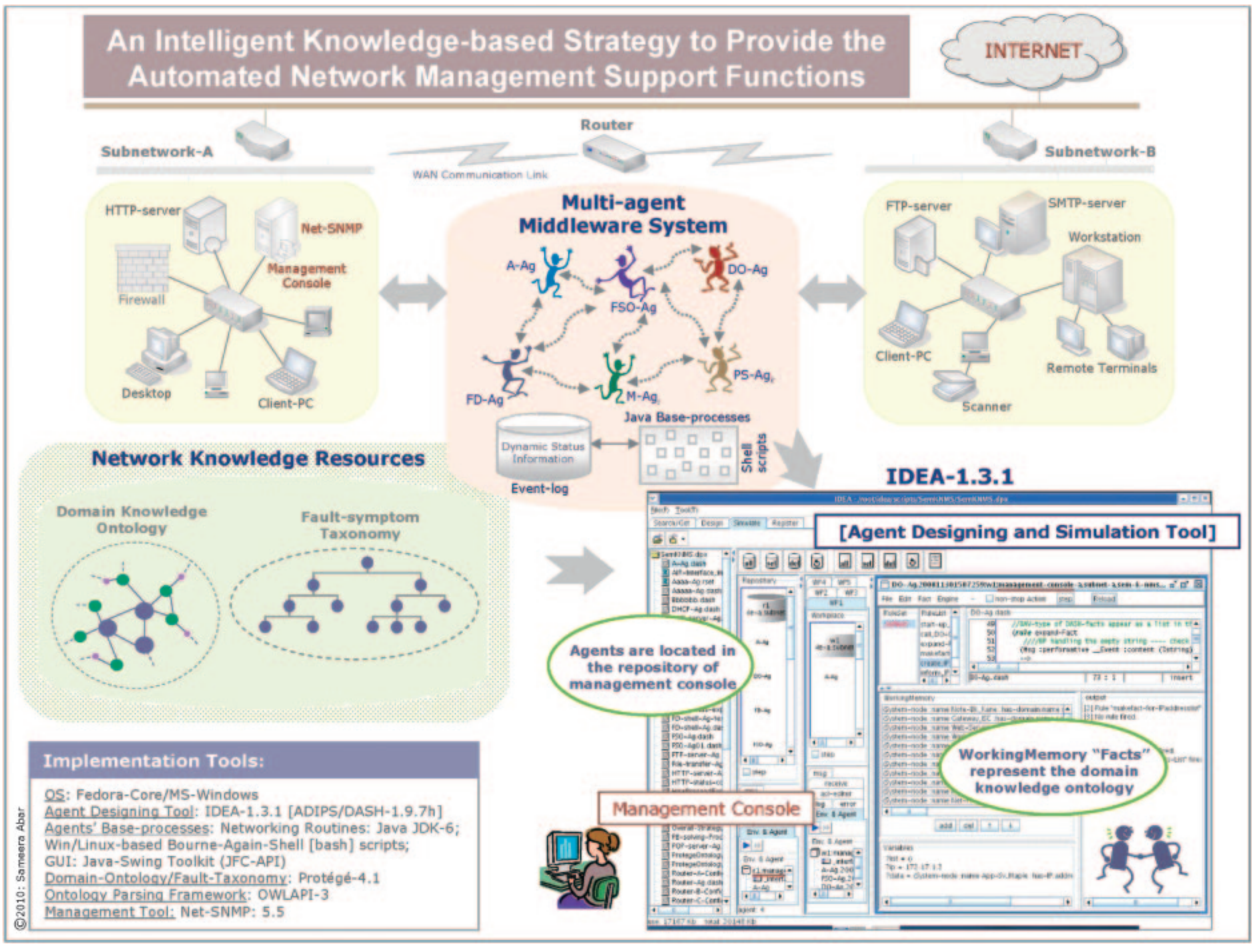

(1) Actual configuration of network elements
(2) MIB Static: Data structure of managed objects in MIB-II
NMD
Static
Dynamic
MIB Variant: Statistical data or numerical facts about operational status of network devices, error reports, server-logs etc.
NMK
Inference
Task
Descriptions of activities to be performed hierarchically, to achieve the desired goal
Heuristic reasoning strategy defines how domain content and structure can be utilized during the diagnostic sessions
• Rules: Logical relations among multiple objects
• Correlatios: Grouping of multiple variables
• Actions: Formal definitions of system behavior – a pair of pre-condition and post-condition

Fig. 1. Network Management Data (NMD) versus Network Management Knowledge (NMK).


An Intelligent Knowledge-based Strategy to Provide the Automated Network Management Support Functions
INTERNET
Router
Subnetwork-A
Subnetwork-B
WAN Communication Link
HTTP-server
Net-SNMP
Management Console
Firewall
Desktop
Client-PC
Multi-agent Middleware System
A-Ag
FSO-Ag
DO-Ag
FD-Ag
M-Ag
PS-Ag
Java Base-processes
Dynamic Status Information
Event-log
Shell scripts
FTP-server
SMTP-server
Workstation
Client-PC
Remote Terminals
Scanner
Network Knowledge Resources
Domain Knowledge Ontology
Fault-symptom Taxonomy
IDEA-1.3.1
[Agent Designing and Simulation Tool]
Agents are located in the repository of management console
WorkingMemory "Facts" represent the domain knowledge ontology
Management Console
©2010: Sameera Abar

Implementation Tools:

OS: Fedora-Core/MS-Windows
Agent Designing Tool: IDEA-1.3.1 [ADIPS/DASH-1.9.7h]
Agents' Base-processes: Networking Routines: Java JDK-6; Win/Linux-based Bourne-Again-Shell [bash] scripts;
GUI: Java-Swing Toolkit (JFC-API)
Domain-Ontology/Fault-Taxonomy: Protégé-4.1
Ontology Parsing Framework: OWLAPI-3
Management Tool: Net-SNMP: 5.5

Fig. 2. Pictorial representation of the proposed knowledge-based strategy for the automated support to network management functions.

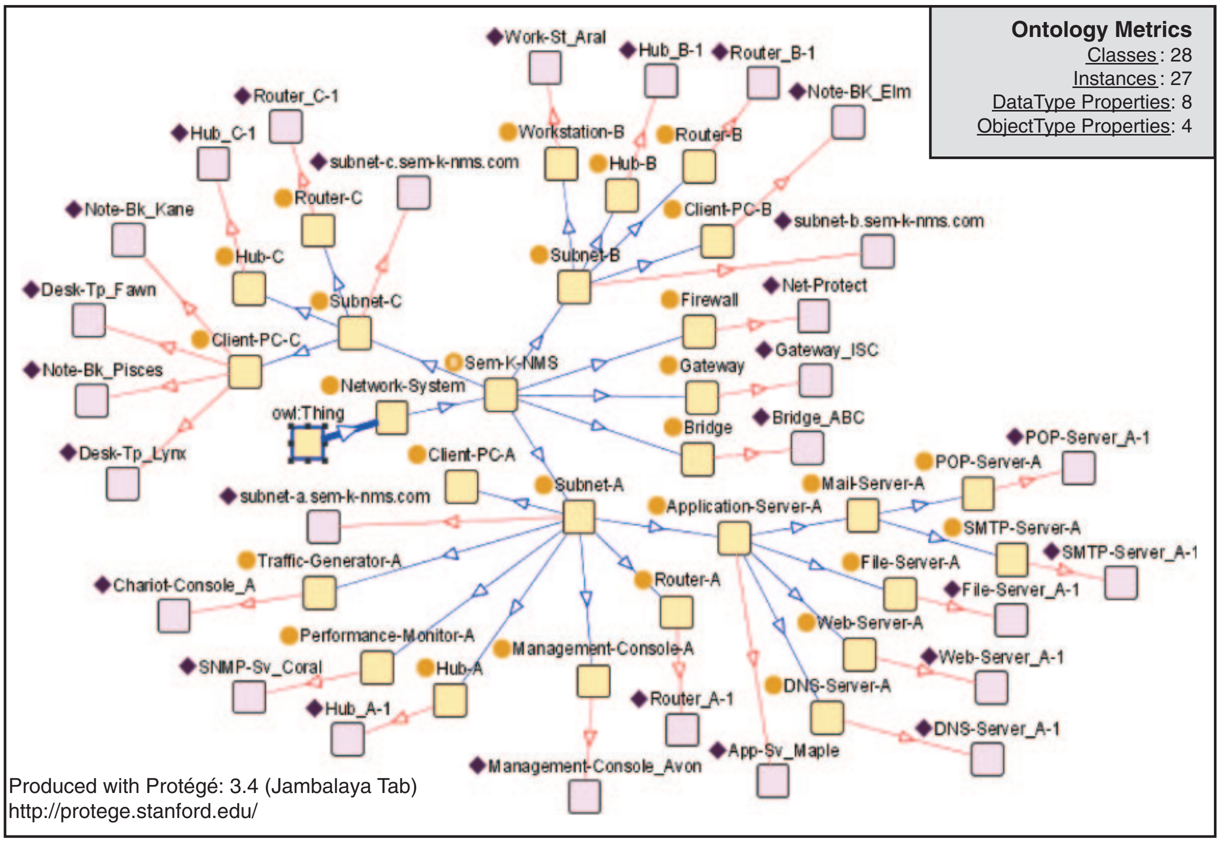
Figure 3 demonstrates our construction of the network systems' domain ontology (DO) using Protégé-3.4 [17], a tool which provides an intuitive graphical user interface for ontology editing and development, a rich knowledge acquisition and modeling features, and an extensible API architecture. The Domain Ontology (DO) in the proposed system is a

Table 1. Proposed framework of the communication networks' knowledge resources.

Network Management Information [CommonKADs]*	Network Management Data		Network Management Knowledge [Procedural]*
	Static Domain Data [Declarative]*	Dynamic Status Data	
Description	**Configurational Specifications** An organized set of domain specific concepts/facts and relationships among them	**Statistical Data or Numerical Facts** Operational status of network devices, error reports, server-logs … Network administrator analyzes this raw data to locate the root cause of the network failures	**Experiential Knowledge** For performing the management tasks **I: Task Knowledge for Fault-diagnosis** [Implicit Design] Hierarchy of Tasks: 1: Symptom Detection 2: Hypothesis Generation 3: Hypothesis Discrimination **II: Inference Support An Agent-based Library of Fault-case Reasoning Models of Network Resources** [Explicit Design] Modeling the cause-effect relations among the occurring faults
Incorporation	**Domain Knowledge Ontology** Nomenclature of network resources, IP-addresses, Port numbers, Routing information, Internet Domain Names, Application settings & versions **[Protégé-3.4]** **[Parser: OWLAPI-2.1.1]**	**Dynamically Generated Run-time Network Data** Collected through RMON (I&II)-MIB or SNMPv2-MIB **[Syslog Functions]**	**Multi-agent Middleware System** Production-rule type knowledge of the DASH-agent describes the behavioral characteristics of faults **[DASH-1.9.7h/IDEA-1.3.1]** Agent-based middleware functions according to the underlying task hierarchy Java-code (as the base-processes of DASH-agents) maps the functionality of empirical knowledge, and provides the inference support with the run-time status data through shell-scripts **[Java-2-SDK-1.6.0]**

[Inference]* Base-processes invoked by the agents, access the run-time dynamic information for reasoning, during the phase of fault-detection and cooperative problem-solving

Experiential knowledge manipulates the domain content for reasoning with the run-time status data



Ontology Metrics
Classes: 28
Instances: 27
DataType Properties: 8
ObjectType Properties: 4

Fig. 3. Standard inheritance view of network systems' domain knowledge ontology.

Protégé-generated OWL-1.0 script (OWL: Web-Ontology-Language is a W3C Standard), and it is parsed and rendered through the OWLAPI-2.1.1 (API: Application Programming Interface) based parser. OWLAPI is a Java-interface, and it subscribes to the axiom-centric view i.e., a set of axioms refer to the entities of an ontology. While designing the network domain ontology for our experimental system, a particular care has been taken to ensure its reusability, maintainability, and modifiability in a flexible manner. The hierarchy of concepts within our experimental network system, and their crucial properties are specified through an OAV (object-attribute-value) mechanism, to make it compatible with the OAV-type data of production rule specifications of software agents. Current version of the domain ontology (DO) embeds the light-weight semantics, and contains 28 classes, 27 instances, and 12 DataType/ObjectType properties.

■ Dynamic Status Information

This is the run-time statistical data collected and analyzed through RMON-MIB (Remote Network Monitoring — Management Information Base) or SNMPv2-MIB (Simple Network Management Protocol version:2 — MIB). The network administrator interprets this raw dynamic status data (event-logs), to locate the root-cause of failures occurring in the network system.

■ Experiential Management Knowledge

— "**Inference Support**" is modeled as FCRMs (Fault-Case Reasoning Models) which depict the behavior of various anomalies occurring in an operational network. There exist some faults in the NMS which are not due to the malfunctioning of physical devices, but occur as a result of software exceptions. The reasoning knowledge prescribes the execution of various task relevant goals in the operational network system, and is incorporated as the FCRMs as shown in Fig. 4. These models depict the behavior of various anomalies occurring in the network objects (physical devices) and resources (software applications), and are based on the observation and experience of the network experts. In the proposed prototype system, the application scenarios of a mail-server, Web-server, and router related anomalies have been designed to offer great flexibility in the diagnosis process.

The libraries of fault-case reasoning models are constructed from a-priori expert knowledge of the probabilities that a specific set of cascading faults, causes the initial symptom. This basically means that we use a causal model of the anomalies to generate those hypotheses that cover or imply all the faulty observations. The cause-effect relationships (a directed acyclic graph where the nodes represent the events and whose directed edges represent causality) among the faults have been explicitly encoded as the production rule-type representation of the DASH (Distributed Agent System based on Hybrid Architecture) [7] agents' knowledge. The FCRMs basically group these small-grain agents to effectively control and reduce the agent messaging traffic, as well as maximize reusability and modifiability.

Moreover, the inference supports are executed on the way of rule invocation. This implies that the agents' rules support inferences during problem-solving, by invoking the underlying Java-based networking routines (base-processes of software agents), upon interaction with the run-time dynamic status information (event-logs generated upon the execution of the Bourne-Again-Shell [bash] scripts), of the operational network system. In addition to this, the automated reasoning tasks with the run-time data are performed by the system agents, in conjunction with the static domain-specific domain ontology, as shown at the bottom-line text of Table 1. Agents' production rule-sets simplify the issuance of rule specification, and enhance the flexibility and provide efficient maintenance of the knowledge-based system. Instantiation of the rules is done when the condition part of the rule is satisfied (after being matched with "Facts" in the agents' Working Memory), it is fired and its action part is executed and consequents are interpreted, by the interaction of Java-based programs with the "syslog" information of the network. Then the agent knowledge modifies dynamically according to the operational characteristics just as MIB's information is retrievable and modifiable in the conventional management solutions.

— "**Task Knowledge Hierarchy**" or problem solving task structures are designed as the generic hierarchy of tasks which prescribe the execution of various activities to be performed in a domain of interest. The diagnosis is defined as the task of identifying the cause of a fault that is manifested by some observed anomalous behavior. The main network diagnosis task for identifying the cause of the occurring fault-symptoms decomposes as the "symptom detection," "hypothesis generation," and "hypothesis discrimination." We refer to these task structures implicitly within the agents' cooperative problem-solving behavior, during the operation of our prototype network system. The Java-programs have also been designed keeping in view the underlying task hierarchy. The above mentioned three sub-tasks are the core of the network management system, and these are the ones for which the knowledge of the agent-based middleware system has been designed. The fault-symptom taxonomy contains the knowledge of nearly all the possible failures, which could be encountered by the network user. This hierarchical-tree consists of fault-symptoms corresponding to the Application, Transport, Network and Data-link layers of the standard OSI/ISO (Open Systems Interconnection/International Standards Organization) model. The fault-symptoms are forwarded in an ordered manner to the fault-detection agent, which determines the actual occurrence of these fault-symptoms in the network system.

Figure 5 shows a flow-chart representing the key functionality of the diagnosis task in our proposed system. When a fault is detected in the experimental network system, then the hypotheses (probable causes of the detected fault) embedded in the fault-case reasoning models about the likely cause of a symptom, are checked by the cooperative problem-solving functions of the multi-agent based middleware system. After the root-cause of the fault-case under consideration has been investigated fully, then another fault-symptom from the fault-symptom taxonomy is forwarded to the system. And this iterative process continues round the clock, for providing the automated fault detection and diagnosis support to network management functions. Figure 6 indicates the interaction and seamless coupling among various knowledge components in our proposed system.

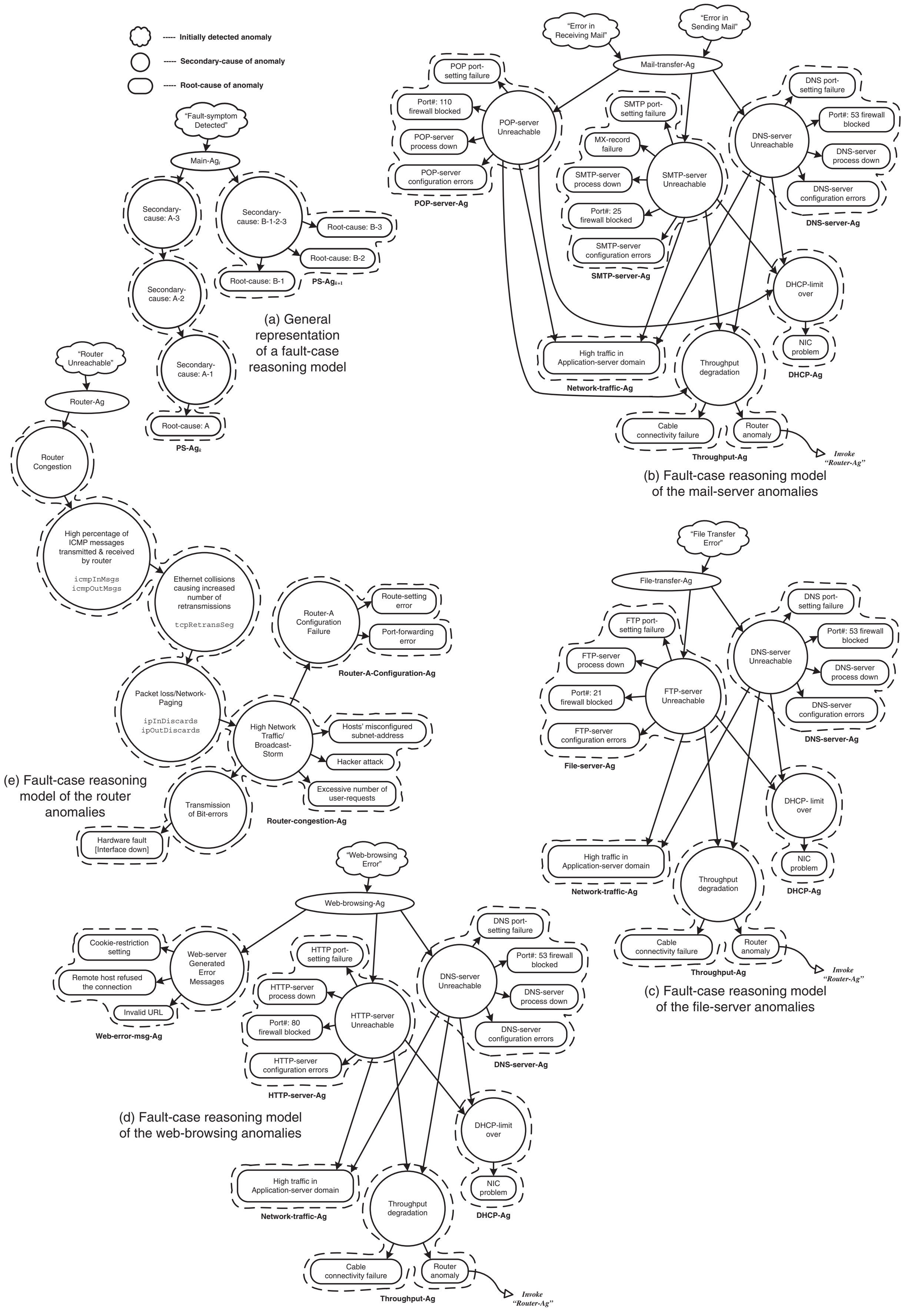

----- Initially detected anomaly
----- Secondary-cause of anomaly
----- Root-cause of anomaly
"Fault-symptom Detected"
Main-Ag$_i$
Secondary-cause: A-3
Secondary-cause: B-1-2-3
Root-cause: B-3
Root-cause: B-2
Root-cause: B-1
PS-Ag$_{k+1}$
Secondary-cause: A-2
Secondary-cause: A-1
Root-cause: A
PS-Ag$_k$
(a) General representation of a fault-case reasoning model
"Error in Receiving Mail"
"Error in Sending Mail"
Mail-transfer-Ag
POP port-setting failure
Port#: 110 firewall blocked
POP-server process down
POP-server configuration errors
POP-server Unreachable
POP-server-Ag
SMTP port-setting failure
MX-record failure
SMTP-server process down
Port#: 25 firewall blocked
SMTP-server configuration errors
SMTP-server Unreachable
SMTP-server-Ag
DNS port-setting failure
Port#: 53 firewall blocked
DNS-server process down
DNS-server configuration errors
DNS-server Unreachable
DNS-server-Ag
DHCP-limit over
NIC problem
DHCP-Ag
High traffic in Application-server domain
Network-traffic-Ag
Throughput degradation
Cable connectivity failure
Router anomaly
Throughput-Ag
Invoke "Router-Ag"
(b) Fault-case reasoning model of the mail-server anomalies
"Router Unreachable"
Router-Ag
Router Congestion
High percentage of ICMP messages transmitted & received by router
icmpInMsgs
icmpOutMsgs
Ethernet collisions causing increased number of retransmissions
tcpRetransSeg
Router-A Configuration Failure
Route-setting error
Port-forwarding error
Router-A-Configuration-Ag
Packet loss/Network-Paging
ipInDiscards
ipOutDiscards
High Network Traffic/ Broadcast-Storm
Hosts' misconfigured subnet-address
Hacker attack
Excessive number of user-requests
Transmission of Bit-errors
Hardware fault [Interface down]
Router-congestion-Ag
(e) Fault-case reasoning model of the router anomalies
"File Transfer Error"
File-transfer-Ag
FTP port-setting failure
FTP-server process down
Port#: 21 firewall blocked
FTP-server configuration errors
FTP-server Unreachable
File-server-Ag
DNS port-setting failure
Port#: 53 firewall blocked
DNS-server process down
DNS-server configuration errors
DNS-server Unreachable
DNS-server-Ag
DHCP- limit over
NIC problem
DHCP-Ag
High traffic in Application-server domain
Network-traffic-Ag
Throughput degradation
Cable connectivity failure
Router anomaly
Throughput-Ag
Invoke "Router-Ag"
(c) Fault-case reasoning model of the file-server anomalies
"Web-browsing Error"
Web-browsing-Ag
Cookie-restriction setting
Remote host refused the connection
Invalid URL
Web-server Generated Error Messages
Web-error-msg-Ag
HTTP port-setting failure
HTTP-server process down
Port#: 80 firewall blocked
HTTP-server configuration errors
HTTP-server Unreachable
HTTP-server-Ag
DNS port-setting failure
Port#: 53 firewall blocked
DNS-server process down
DNS-server configuration errors
DNS-server Unreachable
DNS-server-Ag
DHCP-limit over
NIC problem
DHCP-Ag
High traffic in Application-server domain
Network-traffic-Ag
Throughput degradation
Cable connectivity failure
Router anomaly
Throughput-Ag
Invoke "Router-Ag"
(d) Fault-case reasoning model of the web-browsing anomalies

Fig. 4. Fault-case reasoning models for the probable occurrence of various anomalies in the network system.

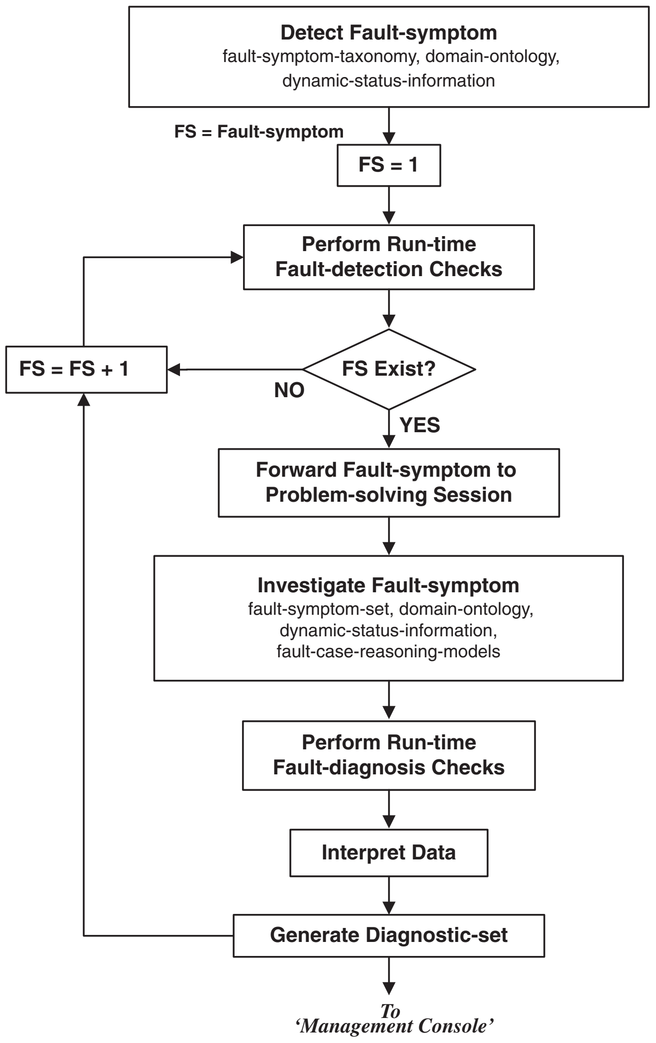

Detect Fault-symptom
fault-symptom-taxonomy, domain-ontology, dynamic-status-information
FS = Fault-symptom
FS = 1
Perform Run-time Fault-detection Checks
FS Exist?
NO
FS = FS + 1
YES
Forward Fault-symptom to Problem-solving Session
Investigate Fault-symptom
fault-symptom-set, domain-ontology, dynamic-status-information, fault-case-reasoning-models
Perform Run-time Fault-diagnosis Checks
Interpret Data
Generate Diagnostic-set
To 'Management Console'

Fig. 5. Flow-chart of fault-management task hierarchy in the proposed system.

3. Implementation Considerations of Multi-agent Middleware

The intelligence layer consists of multi-agents for handling the empirical task-relevant knowledge to relieve the network operators from the tedious monitoring and control of the network system. The proposed system architecture is supported by an Agent-based Distributed Information Processing System (ADIPS) framework (Kinoshita and Sugawara, 1999), which is a flexible computing environment for the implementation of multi-agent systems. The platform chosen for implementing the agent architecture of the proposed prototype system is IDEA-1.3.1 [9] — an Interactive Design Environment for Agent designing framework — ADIPS/DASH-1.9.7h. This provides a repository-based multi-agent computing infrastructure that includes the agent model, development tools, rule-type agent description language, protocols, agent simulation (testing/debugging), and execution environment. The ADIPS framework consists of three sub-systems: "DASH Repository" is a mechanism to manage, organize, and utilize the reusable agents; "DASH Workplace" is an operational environment where the agents are deployed for cooperative problem solving; "DASH Design Support" for designing and implementing the multi-agent middleware system. The repository agents placed on a server in the distributed network system are instantiated onto the workplaces of various computer nodes. DASH framework provides reuse-based design support for constructing the multi-agent middleware layer. The software agent system designers can select suitable agents according to the requirement of services and functions to construct the system in an open and distributed environment. A novel characteristic of the multi-agent system created by the ADIPS/DASH-1.9.7h framework, is the flexible adaptation of its services to the dynamically changing distributed environment, according to the users' requirements as well as the QoS (Quality of Service).

Numerous multi-agent based applications using IDEA have been developed in our laboratory for quite a long time in the context of various academic projects, and the research papers with promising results have been published in the accredited international journals and conferences. The ADIPS/DASH-1.9.7h framework provides the agent developers a mechanism to create an agent and wrap it with a Java process (base-process). The DASH-agent (running on a virtual

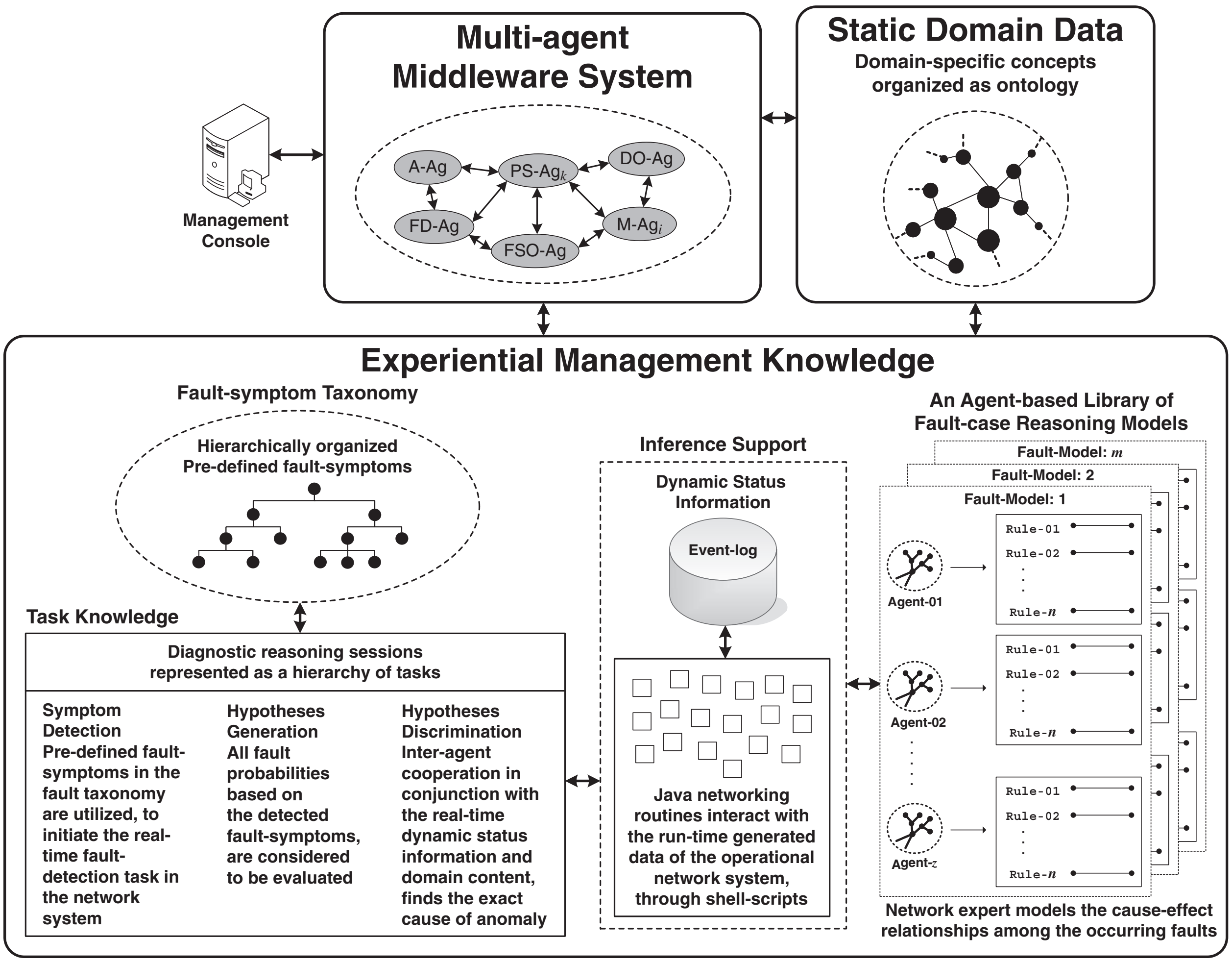

Management Console
Multi-agent Middleware System
A-Ag
PS-Ag$_k$
DO-Ag
FD-Ag
FSO-Ag
M-Ag$_i$
Static Domain Data
Domain-specific concepts organized as ontology
Experiential Management Knowledge
Fault-symptom Taxonomy
Hierarchically organized Pre-defined fault-symptoms
Task Knowledge
Diagnostic reasoning sessions represented as a hierarchy of tasks
Symptom Detection Pre-defined fault-symptoms in the fault taxonomy are utilized, to initiate the real-time fault-detection task in the network system
Hypotheses Generation All fault probabilities based on the detected fault-symptoms, are considered to be evaluated
Hypotheses Discrimination Inter-agent cooperation in conjunction with the real-time dynamic status information and domain content, finds the exact cause of anomaly
Inference Support
Dynamic Status Information
Event-log
Java networking routines interact with the run-time generated data of the operational network system, through shell-scripts
An Agent-based Library of Fault-case Reasoning Models
Fault-Model: m
Fault-Model: 2
Fault-Model: 1
Agent-01
Agent-02
Agent-z
Rule-01
Rule-02
Rule-n
Network expert models the cause-effect relationships among the occurring faults

Fig. 6. Functional representation of the communication networks' knowledge resources.

machine) controls and monitors the base-process running on a distributed computing platform. Java-2-SDK-1.6.0 has been utilized to create the networking routines as the base-processes of DASH-agents, which work in conjunction with their corresponding Linux-based Bourne-Again-Shell [bash] scripts (for the command-line interpreter) and their corresponding output generated on executing the bash-scripts. The GUI (Graphical User Interface) of the management console has been constructed using Java-Swing Toolkit (JFC-API).

3.1 Design Specifications of Multi-agent System

Key role of the agents in our system is to actualize the autonomous fault monitoring and detection function. The internal state knowledge of the system agents comprises of five vital ingredients:

[ID-K] **I**dentification **K**nowledge specifies the environment or the system where the workplace agents are instantiated.

[FD-K] **F**ault-**D**etection **K**nowledge is expressed as the production rule-type instructions in an agent. FD-K inevitability constitutes the core knowledge for detecting the anomalies. For each initial fault symptom reported, various run-time checks performed, and the information obtained (about the occurring obstacle) is compared with the pre-defined error string, and then the result is forwarded to the GUI of the management console.

[PS-K] **P**roblem-**S**olving **K**nowledge is expressed as the production rule-type instructions in an agent. PS-K inevitability constitutes the core knowledge for diagnosing the anomalies. For each initial fault symptom detected, various run-time checks performed for all probable occurrences of the fault-symptom, and the information obtained (about the root-cause of the obstacle) is compared with the pre-defined error string, and then the problem-solving findings are forwarded to the GUI of the management console.

[R-K] **R**esource **K**nowledge specifies the structure and syntax of the diverse range of information required during the agents' cooperative problem solving phases. For instance, the run-time log-information acquired through the "syslog" (a standard logging solution for Unix and Linux systems), and the associated output generated as text-files. R-K also comprises of the faults-related hierarchical knowledge, and the static domain ontological data expressed as "Facts" in the Working Memory of a DASH-agent.

[CM] **C**ooperation **M**echanism represents the protocol sequences of inter-agent cooperation/communication sessions as illustrated in Fig. 7 and Table 2.

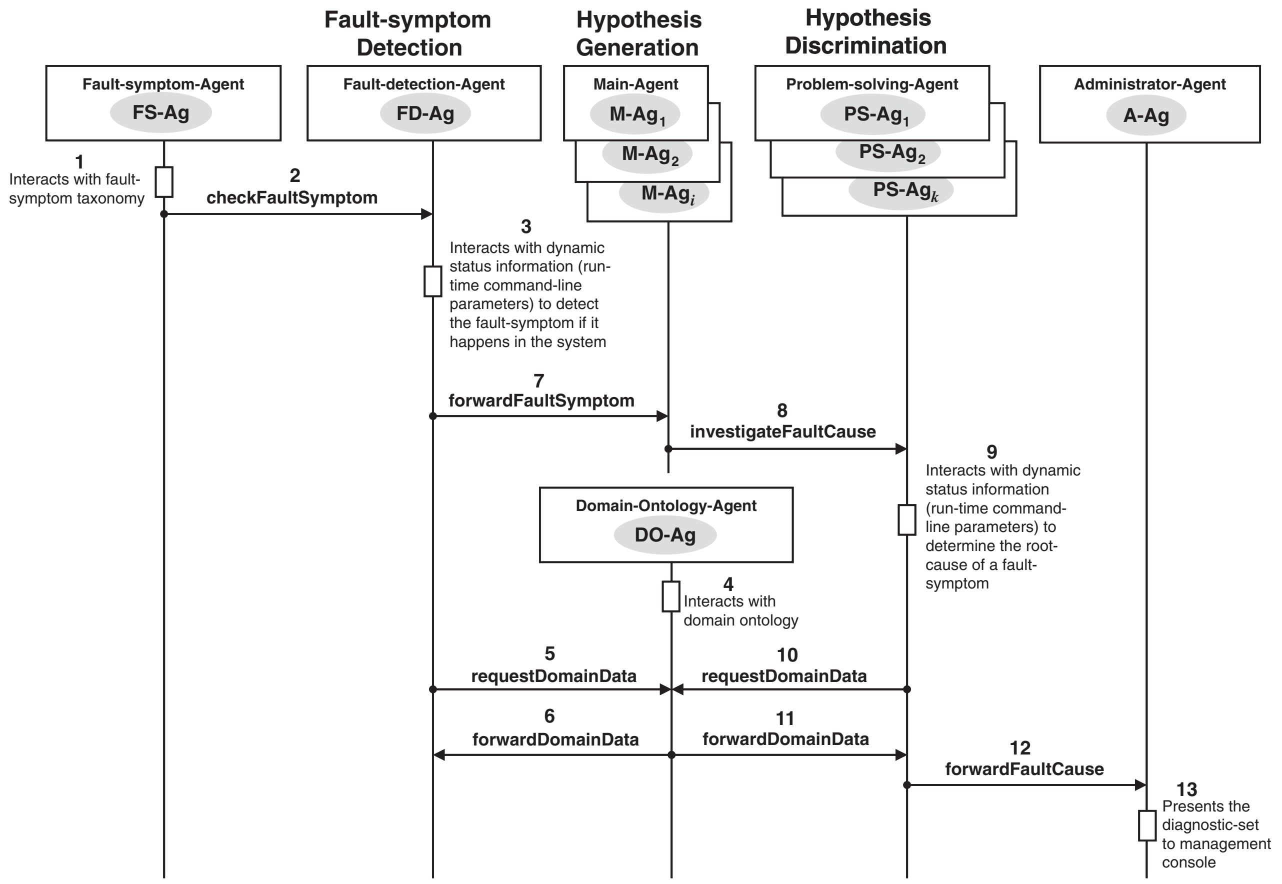

Fault-symptom Detection
Hypothesis Generation
Hypothesis Discrimination
Fault-symptom-Agent
FS-Ag
Fault-detection-Agent
FD-Ag
Main-Agent
M-Ag$_1$
M-Ag$_2$
M-Ag$_i$
Problem-solving-Agent
PS-Ag$_1$
PS-Ag$_2$
PS-Ag$_k$
Administrator-Agent
A-Ag
1 Interacts with fault-symptom taxonomy
2 checkFaultSymptom
3 Interacts with dynamic status information (run-time command-line parameters) to detect the fault-symptom if it happens in the system
7 forwardFaultSymptom
8 investigateFaultCause
9 Interacts with dynamic status information (run-time command-line parameters) to determine the root-cause of a fault-symptom
Domain-Ontology-Agent
DO-Ag
4 Interacts with domain ontology
5 requestDomainData
10 requestDomainData
6 forwardDomainData
11 forwardDomainData
12 forwardFaultCause
13 Presents the diagnostic-set to management console

Fig. 7. Communication/cooperation protocol-sequences of the multi-agent middleware system.

3.2 Functional Behavior of Agents

The functionality of an agent is about how to analyze and process the users' query, as well as defining the cooperation strategy among the multiple systems' agents. The agents embedded in the middleware intelligent support layer, interact with each other cooperatively during the course of network-monitoring and fault-diagnosis sessions. Multi-agent middleware implies that the network diagnostic system can be more scalable, flexible, extensible, and interoperable. Table 3 shows the task specifications of agents in our prototype system. Agents work in conjunction with the dynamic status information (event-logs) of the network system, and the periodic interaction among the agents facilitates the autonomous network management mechanism. Owing to the distributed nature of agents as well as platform independence of Java base-processes, the active utilization of run-time dynamic status information of the large-scale, heterogeneous and distributed environments has become a reality. The functional behavior of our proposed system is explained below:

1: All agents are invoked in the repository of management console as depicted in Fig. 8.

2: On the management console, A-Ag is automatically invoked from the repository to the workplace. Figure 8 also shows the IDEA-1.3.1 Simulator depicting the repository agents instantiated onto the workplace of management console.

3: A-Ag invokes the DO-Ag and FSO-Ag from the repository to the workplace of the management console.

4: DO-Ag (Domain-Ontology Agent) interacts with DO [a Protégé-generated OWL-script], through the OWLAPI-based parser, and fetches the domain ontological concepts in its Working Memory (WM) as shown in Fig. 9.

5: DO-Ag extracts the list of IP-Addresses from the DO and invokes FD-Ag on the workplace of all nodes simultaneously [Fig. 10], to detect the presence of the fault-symptoms occurring in our experimental TCP/IP-based communication network system.

6: On the management console, the GUI (Graphical User Interface) of the prototype system is generated.

7: FSO-Ag interacts with fault-symptom taxonomy [a Protégé-generated OWL-script], through the OWLAPI-based parser. The parsed concepts, attributes and instances of fault-symptom taxonomy then appear as OAV-type list of DASH-Facts in the WM of the FSO-Ag.

8: FSO-Ag forwards these pre-defined fault-symptoms one-by-one automatically to the FD-Ags on all the nodes to detect the probable occurrence of the fault-symptoms in the network system.

9: FD-Ags interact with DO-Ag on the management console, and on need-cum-basis fetch a required list of the parsed

Table 2. Agent communication protocols during the fault-detection and problem-solving sessions.

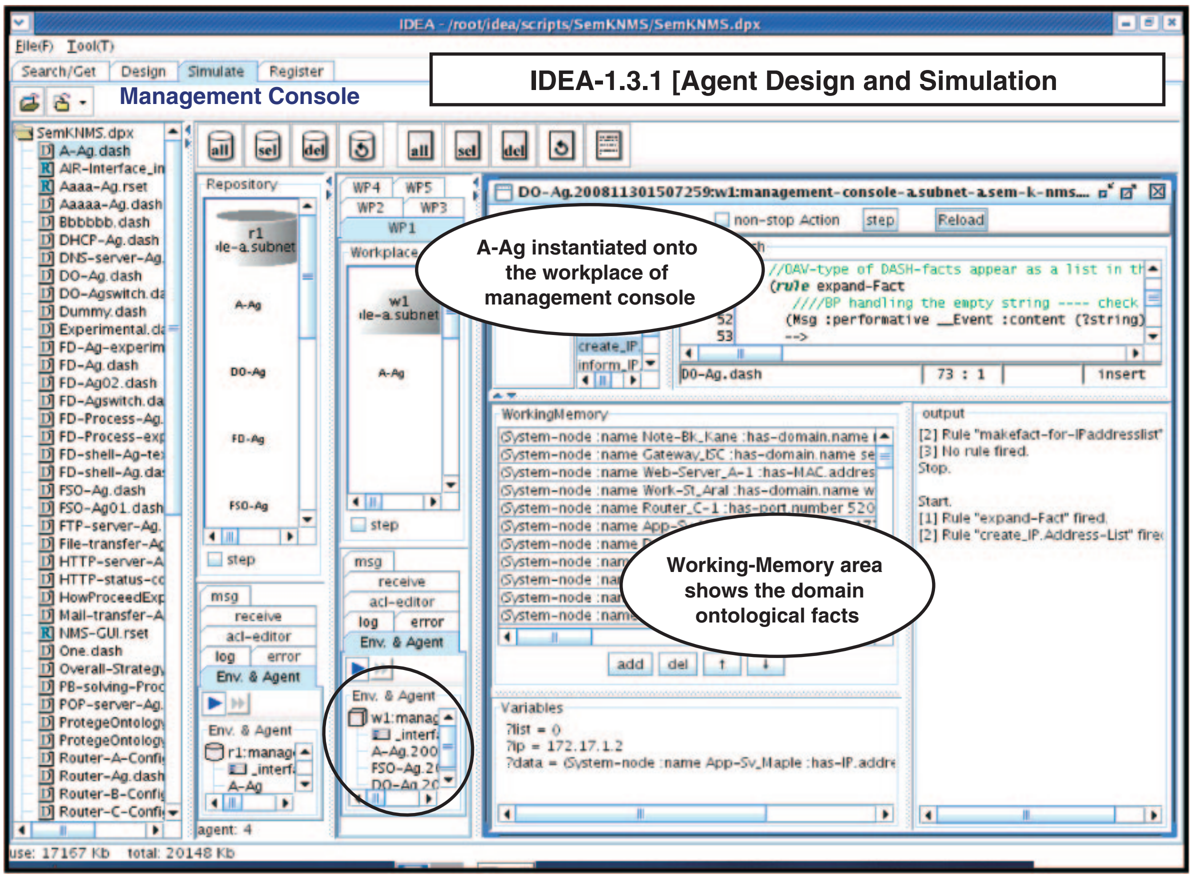
Performative	Function	Remark
checkFaultSymptom	FSO-Ag sends the pre-defined fault-symptom to check for its occurrence, to FD-Ag on all the network nodes	Symptom-detection session starts
forwardFaultSymptom	FD-Ag forwards the detected fault-symptom to the M-Ag$_i$	Symptom-detection session ends
matchFaultSymptom	M-Ag$_i$ matches the detected fault-symptom with the one pre-defined in the fault-case reasoning models	Hypothesis-generation session
investigateFaultCause	M-Ag$_i$ requests the PS-Ag$_k$ to investigate the root cause of failure	Hypothesis-discrimination session starts
requestDomainData	FD-Ag and PS-Ag$_k$ request the DO-Ag for the domain data	Symptom-detection, and Hypothesis-discrimination phases
forwardDomainData	DO-Ag forwards the static domain data to FD-Ag and PS-Ag$_k$	Symptom-detection, and Hypothesis-discrimination phases
forwardFaultCause	PS-Ag$_k$ reports the root cause of failure to A-Ag on the management console	Hypothesis-discrimination session ends

Table 3. Task specifications of agents in the prototype system.

Agent Nomenclature	Task Specification
Fault-symptom-Agent [FS-Ag]	— Interacts with the fault-symptom taxonomy — Gets invoked by the A-Ag after the GUI instantiates onto the management-console's workplace — Forwards the fault-symptom for checking to FD-Ag
Fault-detection-Agent [FD-Ag]	— Receives the request for fault-symptom detection from the FS-Ag — Interacts with the corresponding java-based networking routines — Interacts with DO-Ag for static domain data — Forwards the initially detected fault-symptom to M-Ag$_i$ and A-Ag
Domain-Ontology-Agent [DO-Ag]	— Interacts with the Domain-Ontology (DO) — Receives the request for static ontological data from the FD-Ag and PS-Ag$_k$ — Delivers the static domain data to FD-Ag and PS-Ag$_k$
Main-Agents [M-Ag$_i$]	— Receive the initially detected fault-symptom from FD-Ag — Invoke the corresponding PS-Ag$_k$ — Report the root cause of fault to A-Ag — Intimate the FS-Ag about the completion of fault-diagnosis phase
Problem-solving-Agents [PS-Ag$_k$]	— Start problem-solving upon receiving messages from M-Ag$_i$ — Interact with the corresponding java-based networking routines — Interact with DO-Ag for static domain data — Forward the root cause of fault to M-Ag$_i$
Administrator-Agent [A-Ag]	— Receives the initially detected fault-symptom from FD-Ag, and reports it to GUI of the management console — Receives the results of fault investigation from M-Ag$_i$ and reports it to GUI of the management console

concepts, attributes and instances of DO as the OAV-type list of DASH-Facts. FD-Ags also interact with the corresponding Java networking routines, which in conjunction with the run-time processes or dynamic status data of the network system, locate the initial error occurred in the network system. Figure 11 shows an excerpt of code representing the fault-detection knowledge of an FD-Ag. Shell-scripts (for the conventional network management commands (netstat, ping, ipconfig, route, arp, ps, traceroute, tcpdump, nslookup etc.) are called by the agents in the system, and the output generated by the shell-scripts is directed to create text-files, which are checked for the specific pre-defined error-strings by the agents' Java base-processes.

10: If a fault symptom has been detected by the FD-Ag on any machines (there could be more than one fault-symptoms happening in the network system at any given instance of time), then it is reported to the GUI of the prototype management console.


IDEA - /root/idea/scripts/SemKNMS/SemKNMS.dpx
IDEA-1.3.1 [Agent Design and Simulation
Management Console
A-Ag instantiated onto the workplace of management console
Working-Memory area shows the domain ontological facts
Repository
Workplace
WP1
WP2
WP3
WP4
WP5
step
msg
receive
acl-editor
log
error
Env. & Agent
agent: 4
non-stop Action
step
Reload
add
del
WorkingMemory
Variables
output

DO-Ag.200811301507259:w1:management-console-a.subnet-a.sem-k-nms...

```
//OAV-type of DASH-facts appear as a list in th
(rule expand-Fact
   ////BP handling the empty string ---- check
  (Msg :performative __Event :content (?string)
  -->
```

DO-Ag.dash 73 : 1 insert

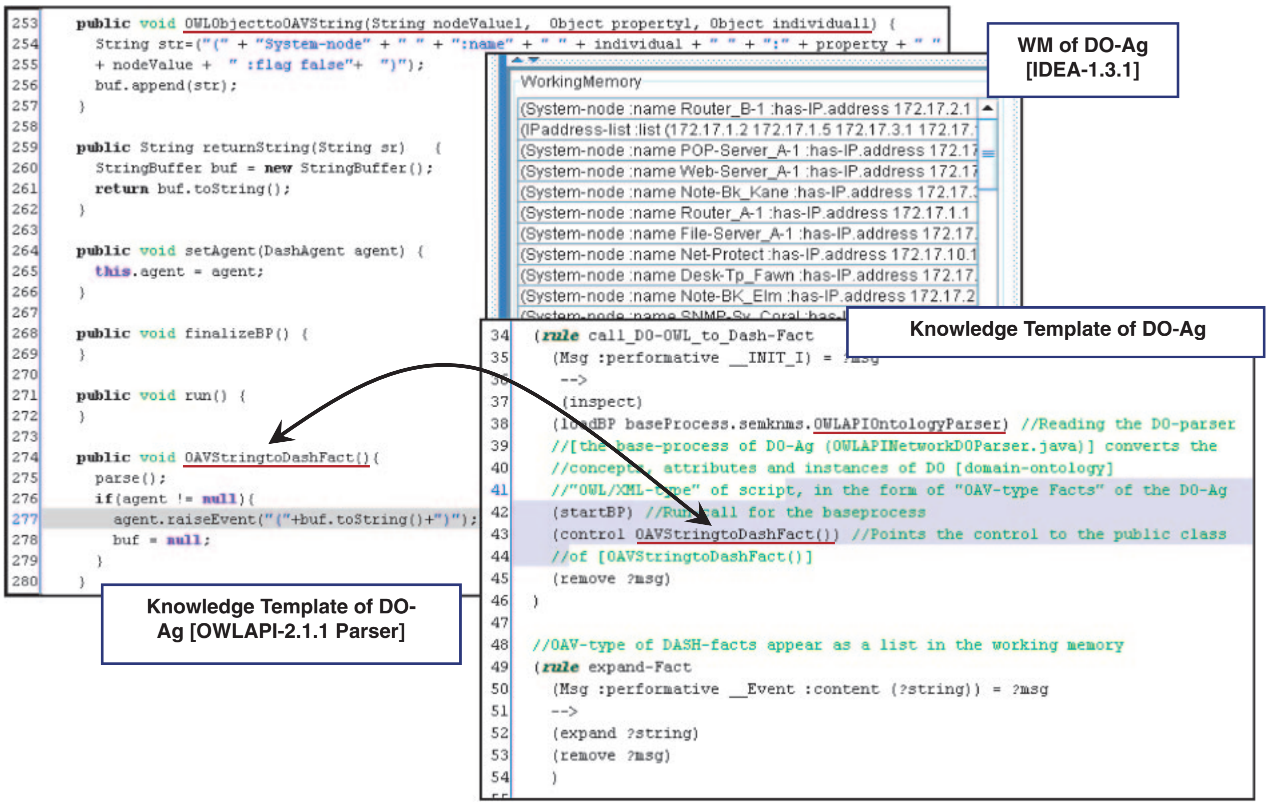
```
(System-node :name Note-Bk_Kane :has-domain.name
(System-node :name Gateway_ISC :has-domain.name se
(System-node :name Web-Server_A-1 :has-MAC.addres
(System-node :name Work-St_Aral :has-domain.name w
(System-node :name Router_C-1 :has-port.number 520
```

```
?list = ()
?ip = 172.17.1.2
?data = (System-node :name App-Sv_Maple :has-IP.addre
```

```
[2] Rule "makefact-for-IPaddresslist"
[3] No rule fired.
Stop.

Start.
[1] Rule "expand-Fact" fired.
[2] Rule "create_IP.Address-List" firec
```

use: 17167 Kb total: 20148 Kb

Fig. 8. IDEA-1.3.1 Simulator depicting repository agents instantiated onto the workplace of management console.

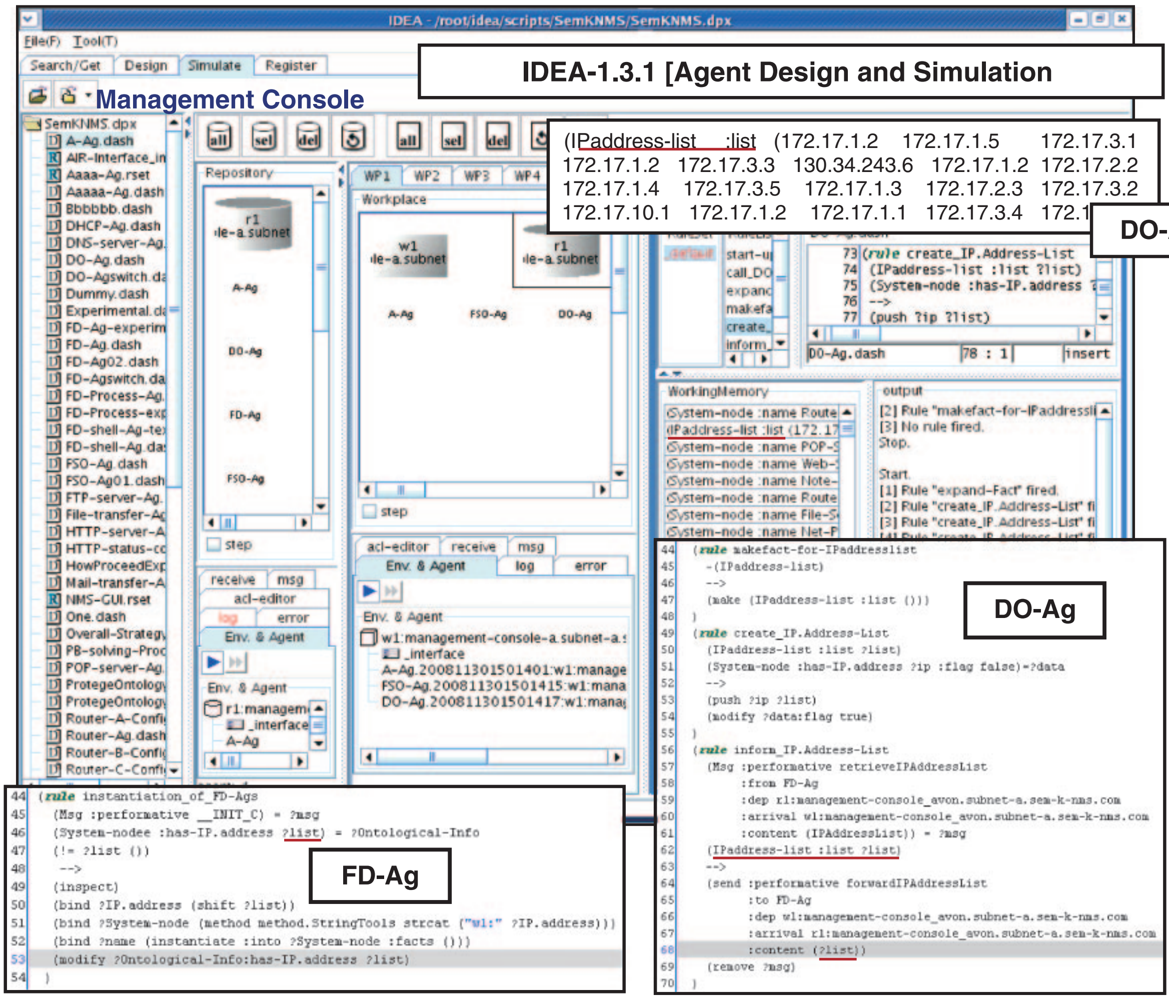

WM of DO-Ag [IDEA-1.3.1]
Knowledge Template of DO-Ag
Knowledge Template of DO-Ag [OWLAPI-2.1.1 Parser]
WorkingMemory

```
    public void OWLObjecttoOAVString(String nodeValuel,  Object propertyl, Object individuall) {
      String str=("(" + "System-node" + " " + ":name" + " " + individual + " " + ":" + property + " "
      + nodeValue +  " :flag false"+  ")");
      buf.append(str);
    }

    public String returnString(String sr)    {
      StringBuffer buf = new StringBuffer();
      return buf.toString();
    }

    public void setAgent(DashAgent agent) {
      this.agent = agent;
    }

    public void finalizeBP() {
    }

    public void run() {
    }

    public void OAVStringtoDashFact(){
      parse();
      if(agent != null){
        agent.raiseEvent("("+buf.toString()+")");
        buf = null;
      }
    }
```

```
(System-node :name Router_B-1 :has-IP.address 172.17.2.1
(IPaddress-list :list (172.17.1.2 172.17.1.5 172.17.3.1 172.17.
(System-node :name POP-Server_A-1 :has-IP.address 172.17
(System-node :name Web-Server_A-1 :has-IP.address 172.17
(System-node :name Note-Bk_Kane :has-IP.address 172.17.
(System-node :name Router_A-1 :has-IP.address 172.17.1.1
(System-node :name File-Server_A-1 :has-IP.address 172.17.
(System-node :name Net-Protect :has-IP.address 172.17.10.1
(System-node :name Desk-Tp_Fawn :has-IP.address 172.17.
(System-node :name Note-BK_Elm :has-IP.address 172.17.2
```

```
 (rule call_DO-OWL_to_Dash-Fact
   (Msg :performative __INIT_I) = ?msg
    -->
    (inspect)
   (loadBP baseProcess.semknms.OWLAPIOntologyParser) //Reading the DO-parser
   //[the base-process of DO-Ag (OWLAPINetworkDOParser.java)] converts the
   //concepts, attributes and instances of DO [domain-ontology]
   //"OWL/XML-type" of script, in the form of "OAV-type Facts" of the DO-Ag
   (startBP) //Run call for the baseprocess
   (control OAVStringtoDashFact()) //Points the control to the public class
   //of [OAVStringtoDashFact()]
   (remove ?msg)
 )

//OAV-type of DASH-facts appear as a list in the working memory
(rule expand-Fact
   (Msg :performative __Event :content (?string)) = ?msg
   -->
   (expand ?string)
   (remove ?msg)
   )
```

Fig. 9. Domain-Ontology Agent (DO-Ag) fetching the ontological concepts in its Working Memory (WM).


IDEA - /root/idea/scripts/SemKNMS/SemKNMS.dpx
IDEA-1.3.1 [Agent Design and Simulation
Management Console
DO-Ag
DO-Ag
FD-Ag

```
(IPaddress-list   :list  (172.17.1.2   172.17.1.5      172.17.3.1
172.17.1.2  172.17.3.3  130.34.243.6  172.17.1.2  172.17.2.2
172.17.1.4   172.17.3.5   172.17.1.3   172.17.2.3  172.17.3.2
172.17.10.1   172.17.1.2   172.17.1.1   172.17.3.4  172.1
```

```
(rule create_IP.Address-List
 (IPaddress-list :list ?list)
 (System-node :has-IP.address ?
 -->
 (push ?ip ?list)
```

```
Start.
[1] Rule "expand-Fact" fired.
[2] Rule "create_IP.Address-List" fi
[3] Rule "create_IP.Address-List" fi
```

```
(rule makefact-for-IPaddresslist
  -(IPaddress-list)
  -->
  (make (IPaddress-list :list ()))
)
(rule create_IP.Address-List
  (IPaddress-list :list ?list)
  (System-node :has-IP.address ?ip :flag false)=?data
  -->
  (push ?ip ?list)
  (modify ?data:flag true)
)
(rule inform_IP.Address-List
  (Msg :performative retrieveIPAddressList
       :from FD-Ag
       :dep r1:management-console_avon.subnet-a.sem-k-nms.com
       :arrival w1:management-console_avon.subnet-a.sem-k-nms.com
       :content (IPAddressList)) = ?msg
  (IPaddress-list :list ?list)
  -->
  (send :performative forwardIPAddressList
        :to FD-Ag
        :dep w1:management-console_avon.subnet-a.sem-k-nms.com
        :arrival r1:management-console_avon.subnet-a.sem-k-nms.com
        :content (?list))
  (remove ?msg)
)
```

```
(rule instantiation_of_FD-Ags
  (Msg :performative __INIT_C) = ?msg
  (System-nodee :has-IP.address ?list) = ?Ontological-Info
  (!= ?list ())
   -->
  (inspect)
  (bind ?IP.address (shift ?list))
  (bind ?System-node (method method.StringTools strcat ("w1:" ?IP.address)))
  (bind ?name (instantiate :into ?System-node :facts ()))
  (modify ?Ontological-Info:has-IP.address ?list)
 )
```

Fig. 10. Domain ontology agent (DO-Ag) forwarding the list of IP-Addresses to fault-detection agents (FD-Ags).

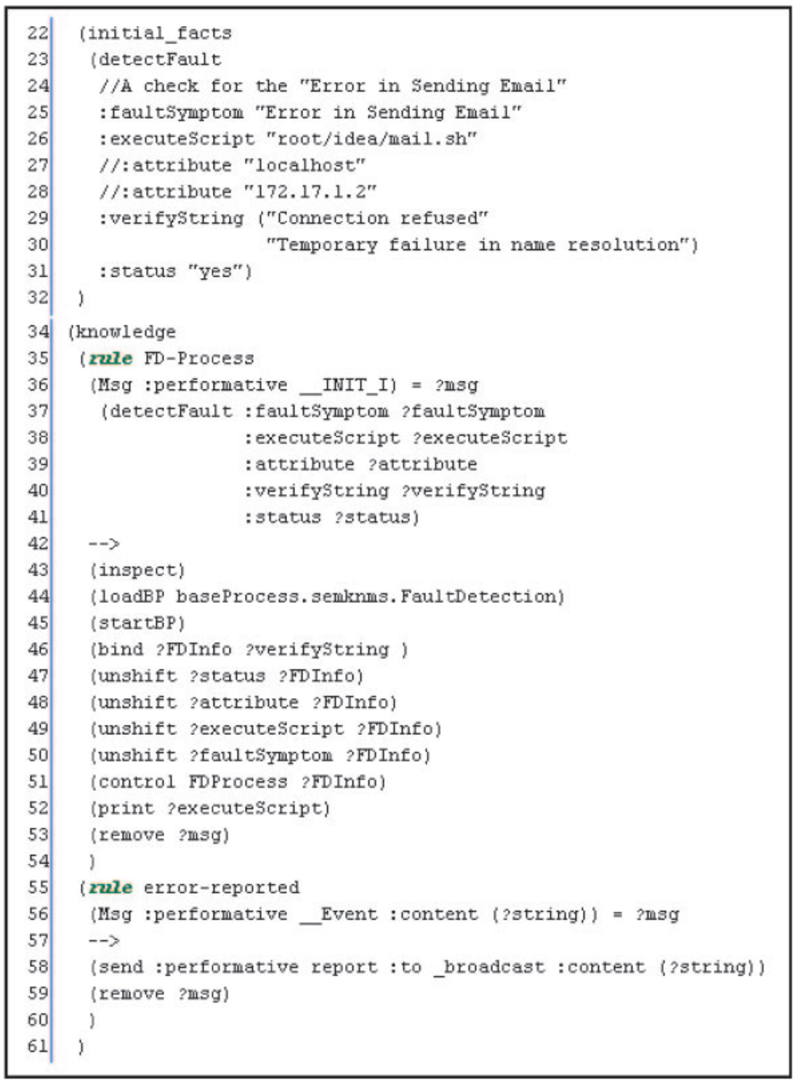
11: FD-Ag sends the detected fault-symptom to Main-Ags (Mail-transfer-Ag, File-transfer-Ag, Router-Ag, and Web-browsing-Ag) on the management console. M-Ags match the detected fault-symptom with the one pre-defined in their fault-case reasoning models. The purpose of Main-Ags is to effectively control and reduce the agent messaging traffic, thereby forwarding the detected fault-symptom to only the relevant problem-solving agents (PS-Ags) for investigating the root-cause of the problem.

12: PS-Ags have been designed to perform checks for a testable hypothesis (an optimal order for a set of tests is determined with the management expertise and encoded as agent-based fault-case reasoning models) on a specific machine. PS-Ags interact with DO-Ag on the management console (upon need-cum-basis), and fetch the parsed concepts, attributes and instances of DO as the OAV-type of DASH-Facts in the WM [Fig. 12]. PS-Ags also interact with the corresponding Java-based networking routines, which in conjunction with the run-time dynamic status data, locate the root cause of initial fault-symptom in the network system. The process of small-grain PS-Ags interacting with Java base-processes has been depicted in Fig. 13.

13: Findings (hypothesis generation phase) of the problem-solving agents are forwarded to the A-Ag residing on the management console for display on the GUI. Once the agents cooperate and interact with each other during the complex problem-solving sessions to resolve a particular failure condition, a part of their embedded knowledge modifies or updates dynamically, according to the run-time conditions of the network system. Then the aggregated root-cause response is reported to the management console thereby restarting the fault monitoring, detection, and diagnosis processes.

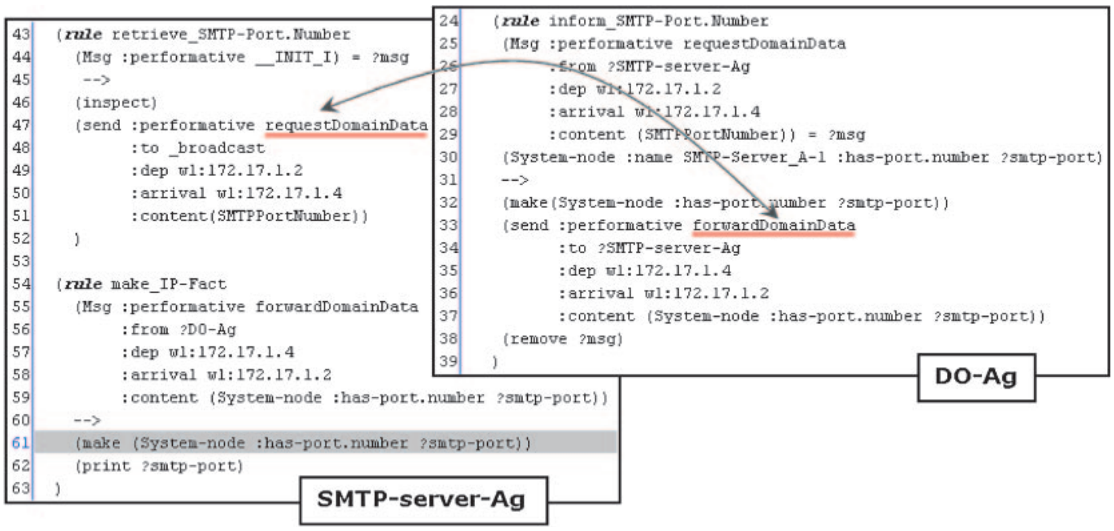
4. Test Environment

Our experimental network system comprises of 100BASE-TX Ethernet, firewall configured as NAT (Network Address Translation), routers, bridges, various personal computers arranged in three Subnetworks (A, B, C) [Fig. 14]. Subnetwork-A is configured as DMZ (DeMilitarized Zone) with dynamic IP-address range 172.17.1.0/24. The application server (HTTP, SMTP, POP, FTP, DNS), the SNMP-server, and the Chariot traffic generator, and the



```
 (initial_facts
  (detectFault
   //A check for the "Error in Sending Email"
   :faultSymptom "Error in Sending Email"
   :executeScript "root/idea/mail.sh"
   //:attribute "localhost"
   //:attribute "172.17.1.2"
   :verifyString ("Connection refused"
                  "Temporary failure in name resolution")
   :status "yes")
 )
(knowledge
 (rule FD-Process
  (Msg :performative __INIT_I) = ?msg
   (detectFault :faultSymptom ?faultSymptom
                :executeScript ?executeScript
                :attribute ?attribute
                :verifyString ?verifyString
                :status ?status)
  -->
  (inspect)
  (loadBP baseProcess.semknms.FaultDetection)
  (startBP)
  (bind ?FDInfo ?verifyString )
  (unshift ?status ?FDInfo)
  (unshift ?attribute ?FDInfo)
  (unshift ?executeScript ?FDInfo)
  (unshift ?faultSymptom ?FDInfo)
  (control FDProcess ?FDInfo)
  (print ?executeScript)
  (remove ?msg)
  )
 (rule error-reported
  (Msg :performative __Event :content (?string)) = ?msg
  -->
  (send :performative report :to _broadcast :content (?string))
  (remove ?msg)
  )
 )
```

Fig. 11. Code-snippet representing the fault-detection knowledge of FD-Ag.



```
(rule retrieve_SMTP-Port.Number
  (Msg :performative __INIT_I) = ?msg
   -->
  (inspect)
  (send :performative requestDomainData
        :to _broadcast
        :dep wl:172.17.1.2
        :arrival wl:172.17.1.4
        :content(SMTPPortNumber))
  )

(rule make_IP-Fact
  (Msg :performative forwardDomainData
       :from ?DO-Ag
       :dep wl:172.17.1.4
       :arrival wl:172.17.1.2
       :content (System-node :has-port.number ?smtp-port))
  -->
  (make (System-node :has-port.number ?smtp-port))
  (print ?smtp-port)
)
```

```
 (rule inform_SMTP-Port.Number
  (Msg :performative requestDomainData
       :from ?SMTP-server-Ag
       :dep wl:172.17.1.2
       :arrival wl:172.17.1.4
       :content (SMTPPortNumber)) = ?msg
  (System-node :name SMTP-Server_A-1 :has-port.number ?smtp-port)
  -->
  (make(System-node :has-port.number ?smtp-port))
  (send :performative forwardDomainData
        :to ?SMTP-server-Ag
        :dep wl:172.17.1.4
        :arrival wl:172.17.1.2
        :content (System-node :has-port.number ?smtp-port))
  (remove ?msg)
 )
```

Fig. 12. DO-Ag forwarding the static domain-content to the problem-solving agent (SMTP-server-Ag).

network management console resides in the Subnetwork-A. The workstation and a notebook computer system constitute the Subnetwork-B, and the nodes of Subnetwork-C comprise of three notebook computers and a desktop system. Moreover, on each node Fedora Core:ver-2.0 has been installed as an OS (Operating System) for the machines.

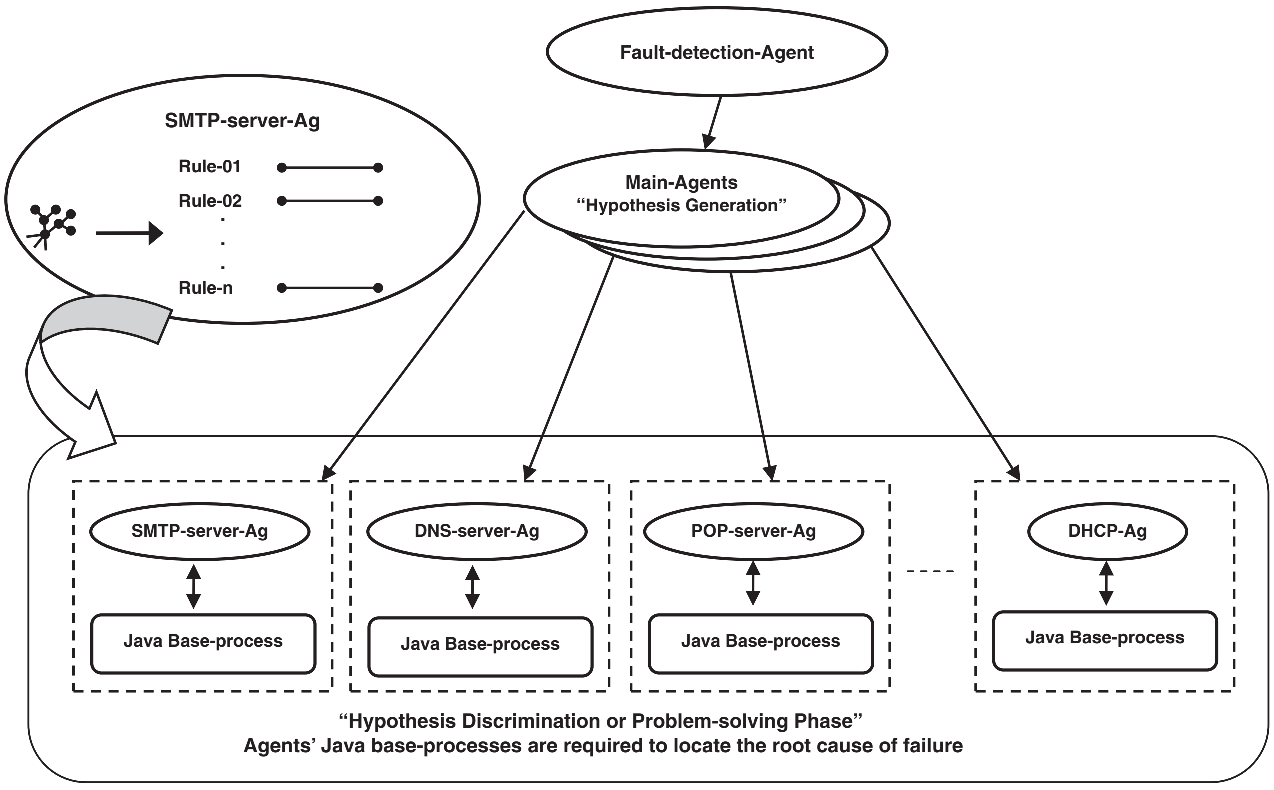


Fig. 13. Small-grain PS-Ags agents interacting with Java base-processes.

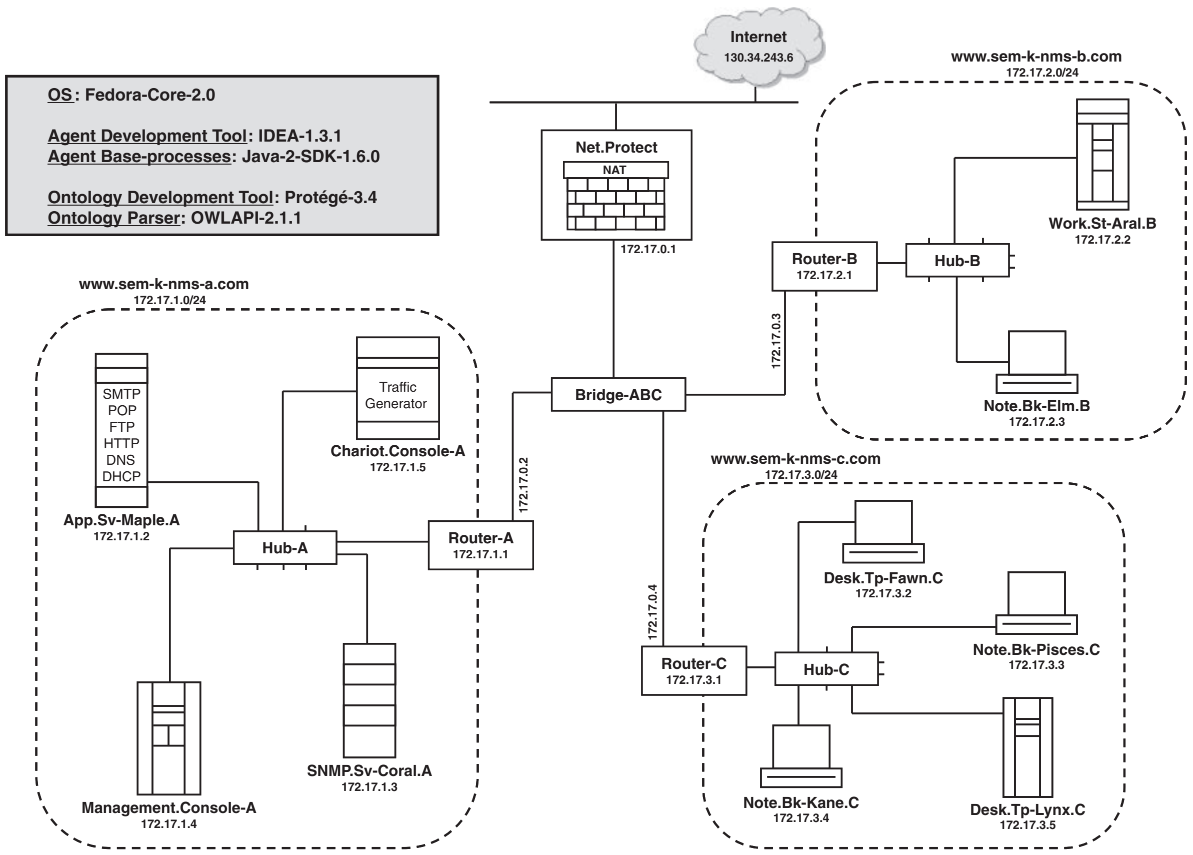


Fig. 14. Configuration of the test network system.

The front-end GUI (Graphical User Interface) of the management console has been created using Java Swing-toolkit. This test-bed system facilitates the automated fault monitoring, detection, and diagnosis sessions that can run unattended all the time, thereby maximizing its utilization for supporting the network management tasks.

Table 4. Application scenarios for experimenting with the prototype system.

Test Case	Network Node (Machine)	Initially Detected Fault-symptom	Root-cause of Failure
1	**Router_A-1**	**Network-paging [Slow-speed]**	**Excessive User-requests or Connectivity Failure**
	Performance-degradation occurs due to data bursts caused from the excessive number of user-requests, or equipment/link failure		
2	**Work-St_Aral**	**FTP-server Access Denied**	**Power Failure in Router-A_1**
	Power failure in Router_A-1 results in broken connectivity. Error messages from the data-transport layer are generated for all affected connections. As a consequence, disruption in the file access occurs		
3	**Note-Bk_Elm**	**Error in Sending Email**	**SMTP-server Process Down**
	SMTP-server process "sendmail" goes down in the application server [App-Sv_Maple], thereby causing an error while sending the email		
4	**Desk-Tp_Fawn**	**Host Connection Refused**	**HTTP-server Port Firewall-Blocked**
	Port of HTTP-server in Subnet-A is blocked by the firewall [ip-tables], causing the connection refused error or an error-page is displayed		
5	**Note-Bk_Pisces**	**Error in Receiving Email**	**POP-server Port-setting Failure**
	POP-server port has been configured incorrectly in the application server [App-Sv_Maple], thereby causing an error in receiving the email		
6	**Desk-Tp_Lynx**	**Destination Host Unreachable**	**Incorrect NIC-setting**
	Due to the NIC-setting problem of Desk-Tp_Lynx, connection is refused to HTTP-server in Subnet-A		
7	**Note-Bk_Kane**	**Web-browsing Error**	**DNS Configuration Problem**
	DNS misconfiguration results in the failed domain name resolution, hence Web-browsing is inaccessible		

4.1 Identification of Application Scenarios

For evaluating the performance of our prototype system, some real-time tests have been designed and compared with the conventional network management tools and mechanisms. Our system monitors and detects the networks' operational anomalies automatically, generates hypotheses and then in conjunction with the operational networks' dynamic status information (incorporated with Java base-processes) locate the root cause of failures, thereby reducing the workloads of network administrator remarkably.

A couple of test-cases (related to the commonly occurring network anomalies in our experimental environment) have been designed as depicted in Table 4, to demonstrate the functionality of our prototype system. A typical observation is that the failures in the TCP/IP-based networking environment occurring at the lower-layers ("Data-link" or "Network"), cause malfunctioning at the "Application" level significantly. Therefore, the lower-level failures in the communication network system must be monitored and detected effectively. In addition to this, some other lower-layer related test scenarios have also been modeled; for instance, the throughput bottleneck or high packet loss-rate occurs as a consequence of the workstation or router congestion as well as traffic over-load at the application server. Furthermore, the anomalies such as Web-browsing unavailable, an FTP-related error due to the routers' broken connectivity, DNS/SMTP/POP port-setting or configuration problems etc., have also been taken into account. Web-browsing errors, such as the connection refused or error-page displayed, if the port of the HTTP-server is blocked by the firewall or when the server is down, have also been considered. For the Physical/Data-link layer anomalies, the failure of router or NIC (Network Interface Card) has been considered, due to which the broadcast storms or Ethernet collisions cause an increased number of re-transmissions, thereby reducing the throughput of the network to a minimum. The systems' experimental results confirm a marked reduction in the management-overhead of the network administrator, as compared to the manual network management techniques, in terms of the time-taken (from the fault-symptom detection until its root cause determination) and effort-done (number of procedures adopted) during a particular fault-diagnosis session. The accuracy of network management functions in the proposed system depends on the quality of knowledge modeled in the form of domain ontology, fault-symptom taxonomy, agent-based fault-case reasoning models, Java base-processes and bash-scripts. Experimental results confirm that accuracy of our automated prototype system fully complies with the network management standards.

5. Evaluating the Modifiability/Reusability Aspects of the System Design

Currently, reuse-based software development involves identifying knowledge components reusable in different application domains. The reprocessing of knowledge components for expert systems generally conforms to the following characteristics: modifiability, reusability, and shareability; and same are the design focus of our proposed

Table 5. Reuse of agent-based fault-case reasoning models in the proposed system.

Fault-Case Reasoning Model (FCRM)	Agent	Base-Process	Agent-Rules		Reduction in effort for the reuse of agent rules (%)
			Total	Reused	
FCRM for SMTP-related Anomalies	**5**	**2**	**12**	**0**	**0**
FCRM for POP-related Anomalies	**5**	**2**	**11**	**7**	**63.63**
FCRM for FTP-related Anomalies	**5**	**2**	**11**	**7**	**63.63**
FCRM for HTTP-related Anomalies	**6**	**2**	**14**	**7**	**50**
FCRM for Router-related Anomalies	**2**	**1**	**6**	**2**	**33.34**

networks' knowledge framework. Our modeling approach for the communication networks' knowledge acquisition results in a set of reusable components, such as the networks' domain ontology, fault symptom taxonomy, agent-based fault propagation models, Java base-processes, and bash-scripts of the software agents.

5.1 Modifiability Assessment

The networks' static domain content has been represented as the domain knowledge ontology as well as fault-symptom taxonomy, and it is placed at the centralized management console in our prototype communication network system. Upon the need to scale-up the network system, a considerable reduction in effort has been reported to update the domain knowledge ontology in case of the proposed system. This is due to the reason that the static domain ontology can be modified in an efficient manner, due to being present at a centralized location (management console) in the network system. Otherwise, it would have been required to modify the static domain content on each distributed network node. Similarly, the domain ontology allows an incremental development of the static network content and it is gradually expandable over time without loosing integrity for the altered configuration of the network system (taking into consideration the specific characteristics of the communication network domain). When it is required to update or amend the fault-symptom taxonomy, this can be done very easily by enhancing the fault-symptom list within it, on the centralized management console. Hence, the acquisition, sharing, and utilization of static domain knowledge have been facilitated in the proposed prototype system.

5.2 Reusability Assessment

In the proposed system, the expertise of human networking expert has been declaratively encoded as agents' production rules to facilitate the reusable knowledge acquisition, representation, sharing mechanism. Table 5 demonstrates a considerable reduction in an effort (an average of $\sim$52.65%) while designing the agent-based FCRMs (Fault-Case Reasoning Models), for various application scenarios. The FCRM of SMTP-related failures has been created initially as shown in the Fig. 4(b). When this model was reused for POP-related anomalies, the seven agent-rules were reused without any modification, whereas it was required to create only four fresh rules for the POP-specific problems. The knowledge acquisition and reuse activity has reported that when these small-grain agents were reused for POP, FTP, HTTP and router-related anomalies, a considerable reduction in the design-effort for the reuse of experiential knowledge (agents' production rules of FCRMs) has been confirmed as shown in Fig. 15. More specifically, these pre-existing communication networks' domain-specific libraries of small-grain agents in the system can be reused flexibly across different application scenarios. A knowledge engineer can select the fragments from our library of reusable knowledge components in accordance with the suitability criteria, and combine them instead of constructing strategy from scratch thereby minimizing the development time and maintenance effort and costs. However, it is observed that the Java base-processes or the networking routines associated with the agent programs, are more generic in nature, and can be reused flexibly across various test-cases in the proposed system as well as other practical diagnostic domains.

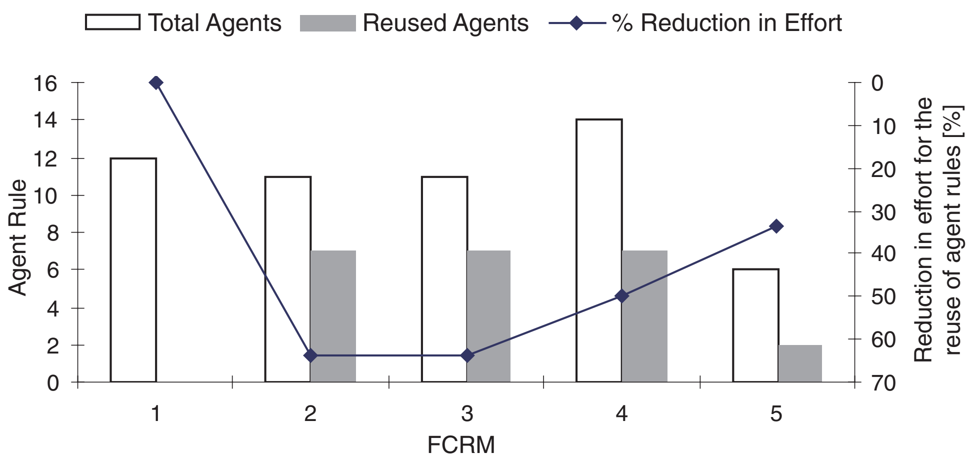


Fig. 15. Reuse of agent-based fault-case reasoning models in the proposed system.

5.3 Analysis — Modifiability/Reusability Aspects of the Proposed Knowledge Model

Estimation of the generic worth of knowledge structures in terms of reusability, maintainability, and modifiability has been performed. In the proposed system, the knowledge items (domain ontology, fault-case propagation models, multi-agents in the middleware layer and the corresponding Java base-processes, bash-scripts) have been designed, constructed, maintained, and refined independently and flexibly. The modifiability/reusability analysis concludes that when some alteration in the network systems' domain content and network management tasks/functions is required to be performed, then it does not reflect any significant changes in the knowledge fragments embedded in the proposed system. The knowledge components are adaptable and scalable and can therefore be appropriately configured and integrated into other application environments in a seamless manner. A part of the content of the pre-existing domain ontology, FCRMs, agent modules, Java base-processes) can be copied to another at design-time and compiled at run-time possibly after some minor revisions and refinements according to the characteristics and requirements of the new environment. It has been reported that the experiential knowledge reuse reduces the design-effort for agent-based FCRMs to approximately an half. This makes the knowledge acquisition activity less complex. However in this analysis, it has been assumed that an equal amount of effort is required to design/upgrade/modify each knowledge component.

6. Concluding Remarks

Key to automated network management lies in leveraging the network systems' knowledge resources to reduce the communication complexity and administrative overhead. The focus point of our work is the elicitation of communication networks' knowledge in a generic and reusable manner, to be applicable during the course of automated performance analysis, and anomaly monitoring and detection in the computer (data) communication networks. We have presented a reuse-based innovative knowledge acquisition, representation, and sharing approach for the automated management support to communication network systems. More specifically, the designed communication networks' resource knowledge module comprises of the static content represented as domain knowledge ontology, and the management expertise are encoded as fault-case causal reasoning models of the network objects, which are realized as the production rule-type knowledge of the software multi-agent based middleware. The agents' rules along with the embedded generic Java-based problem solving algorithms and real-time log information perform the automated management tasks. For the proof of concept, the deployment of some test-bed application scenarios has been done in the test network system. This research project results in a fully functional practical network management system, to provide the automated network management support round the clock, thereby reducing the workloads of the network administrator remarkably. This effort can be regarded as one of the initial steps towards the construction, mapping, and implementation of domain content and fault-case causal reasoning knowledge models, for providing the automated management support to the distributed communication network systems. In a nutshell, this research project serves as an ample resource for engineering the next-generation state of art reuse-based knowledge solutions for the adaptive and autonomic computing in distributed environments.

Acknowledgements

This research has been partially supported by the Japan's MEXT "21st-Century COE (Center Of Excellence) Program [System Construction of Global-Network-Oriented Information Electronics]" and the "Global COE Program

[CERIES]” of Tohoku University (2005–2009), and the “Tohoku Kaihatsu Kinen Foundation,” Japan (2006–2008). Authors are thankful to Dr. Yukio Iwaya, Dr. Takahiro Uchiya, Dr. Susumu Konno, Mr. Takahide Maemura, and Mr. Satoshi Yoshimura for their worthful technical assistance. Authors would like to acknowledge the valuable comments and suggestions of anonymous reviewers.

REFERENCES

[1] Abar, S., Hatori, H., Abe, T., and Kinoshita, T., “Agent-based Knowledge Acquisition in Network Management Domain,” Proceedings of the 19th IEEE International Conference on Advanced Information Networking and Applications (AINA-05), Vol. 1, Taipei, Taiwan, 687–692 (2005).

[2] Abu-Hanna, A., and Jansweijer, W., “Modeling Domain Knowledge Using Explicit Conceptualization,” IEEE Expert, Vol. 9, No. 6, 53–64 (1994).

[3] Bernaras, A., Laresgoiti, I., Bartolome, N., and Corera, J., “Building and Using an Electrical Network Ontology for Fault Diagnosis,” Engineering Intelligent Systems, Vol. 6, 3–11 (1998).

[4] Brewster, C., and O’Hara, K., “Knowledge Representation with Ontologies: The Present and Future,” IEEE Intelligent Systems, Vol. 19, No. 1, 72–73 (2004).

[5] Chandrasekaran, B., Johnson, T. R., and Smith, J. W., “Task Structure Analysis for Knowledge Modeling,” Communications of ACM, Vol. 35, No. 9, 124–137 (1992).

[6] Clancey, W. J., “Heuristic Classification,” Artificial Intelligence, Vol. 27, No. 3, 289–350 (1985).

[7] DASH — Distributed Agent System based on Hybrid architecture. Available Online: http://www.agent-town.com/dash/index.html

[8] Hamdi, M. S., “MASACAD: A Multi-agent based Approach to Information Customization,” IEEE Intelligent Systems, Vol. 21, No. 1, 60–67 (2006).

[9] IDEA — Interactive Design Environment for Agent designing framework Available Online: http://www.ka.riec.tohoku.ac.jp/idea/index.html

[10] Kinoshita, T., and Sugawara, K., “ADIPS Framework for Flexible Distributed Systems,” In: Ishida, T. (ed.): Multiagent Platforms. Lecture Notes in Artificial Intelligence, Vol. 1599, Springer-Verlag, 18–32 (1999).

[11] Konno, S., Iwaya, Y., Abe, T., and Kinoshita, T., “Design of Network Management Support System Based on Active Information Resource,” Proceedings of the 18th IEEE AINA Conference, Fukuoka, Japan, 102–106 (2004).

[12] Lemos, M. A., Barros, L. N., Bernal, V., and Wainer, J., “Building Reusable Knowledge Models for the Communication Network Domain,” Proceedings of the 4th Australian Knowledge Acquisition Workshop (AKAW), Sidney, Australia, 381–390 (1999).

[13] Li, J., and Leon, B. J., “SNMK: Simple Network Management Knowledge,” Proceedings of the 6th IEEE NOM Symposium, Vol. 2, 381–390 (1998).

[14] Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T., and Swartout, W. R., “Enabling Technology for Knowledge Sharing,” AI Magazine, Vol. 12, No. 3, 36–56 (1991).

[15] Noy, F. N., and Musen A. M., “Ontology Versioning in an Ontology Management Framework,” IEEE Intelligent Systems, Vol. 19, No. 4, 6–13 (2004).

[16] O’Leary, D. E., “How Knowledge Reuse Informs Effective System Design and Implementation,” IEEE Intelligent Systems, Vol. 16, No. 1, 44–49 (2001).

[17] Protégé — A Knowledge Acquisition Tool. Available Online: http://protege.stanford.edu/

[18] Schreiber, G., Wielinga, B., Hoog, R., Akkermans, H., and Velda, W. V., “CommonKADS: A Comprehensive Methodology for KBS Development,” IEEE Expert, Vol. 9, No. 6, 28–37 (1994).