An Intelligent Network for Federated Testing of NetCentric Systems

Edward Chow, Mark James, Hsin-Ping Chang, Farrokh Vatan, Gurusham Sudhir
NASA Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109
edward.chow@jpl.nasa.gov

Abstract — *We will describe a dynamic federated autonomic networking system[1] for the testing of netcentric systems across organizations. Using a suite of policy-based management software tools, our system can provide netcentric missions with self-configuring, self-debugging, self-healing, and self-protecting capabilities across end-to-end coalition networks. The novelty of our system originates from: (1) A multi-party dynamic policy negotiation algorithm and protocol for autonomous cross-organization negotiations on network resources; (2) A RDF/XML based language to enable automated understanding of network and application configurations across organizations; (3) A teachable, natural language policy capture system; (4) An advanced user interface tool for capturing and representing domain expert knowledge; and (5) A reasoning system for distributing and applying domain knowledge. This system has been developed for large-scale netcentric systems testing applications where the rapid configuration and management of dynamically changing netcentric test exercises is critical. Using a scenario from a real test exercise involving test ranges around the country, we have demonstrated in our testbed that we could reduce the debugging time for a network configuration problem from 6 hours, involving dozens of operators, to less than 2 seconds with minimal human involvement.*

Index Terms — Policy-Based Network Management, Federated Autonomic Networking, Policy Negotiation, Integration Markup Language, Knowledge Reasoning System.

I. INTRODUCTION

The U.S. Department of Defense (DoD) is moving to a network centric environment to achieve information dominance for military operations. The fielding of netcentric systems requires new, innovative ways to conduct operational testing. The netcentric operational tests are normally achieved by actual operational users executing a series of operationally realistic scenarios while test operators observe and record the activities of the users. This requires test operators to have far reaching access and control of systems often located in geographically distributed locations belonging to different organizations. Because of potential security impacts to these organizations, it is a time consuming process to setup these large scale operational testing of netcentric systems.

The goal of the **P**olicy-Based **A**daptive Network and Security **M**anagement (PAM) Project is to develop the next generation test infrastructure management technology for enabling automated netcentric systems testing across DoD test infrastructures. The PAM technology will address key netcentric system testing challenges such as real-time configuration of mission test events, efficient deployment of cross domain security solutions over test infrastructures, and dynamic management of distributed test infrastructures. The PAM team will apply autonomic networking principles to develop solutions for these challenges. However, the cross-organizational nature of the distributed test infrastructure significantly complicates the potential PAM solutions.

The goal of autonomic networking research is to develop a self-managing network to address growing Internet complexity issues. There are a number of research projects [1] [2] and a few commercial products [3] [4] based on the autonomic networking concept. Most of these technologies focus on autonomic networking within an enterprise. The federation across enterprises is typically done through static arrangements. For large scale military exercises, the static federation processes is very complex and often takes several months to a year. This does not satisfy the critical PAM dynamic infrastructure management requirement.

The available research in dynamic federated autonomic networking is very limited. The Federated, Autonomic Management of End-to-end communication services (FAME) Strategic Research Cluster is developing autonomic management solutions that can be applied to build federated network and service management. A conference presentation [5] was given to define the key challenges. There have been no publications from FAME on end-to-end implementations. The SLA-based SERViceable Metacomputing Environment (SERVME) project [6] focuses on autonomic provisioning and de-provisioning of services according to requesters' Quality of Service (QoS) requirements. However, it does not address the PAM autonomic network management problems.

In this paper, we will describe our implementation of a dynamic federated autonomic networking system, with the organization of the paper as follows. In Section II, we will provide background information on the Jet Propulsion Laboratory (JPL) Policy-Based Management (PBM) software suite. In Section III, we will present our approach and the PAM system architecture. In Section IV, we will describe the PAM system software implementation. In Section V, we will

[1] The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology and was supported in part by the U.S. Department of Defense, Test Resource Management Center, Test & Evaluation/Science and Technology (T&E/S&T) Program under NASA prime contract NAS7-03001, Task Plan Number 81-12346.

978-0-7695-4238-6/10 $26.00 © 2010 IEEE
DOI 10.1109/POLICY.2010.29

present the results of applying our technologies to the netcentric system testing problems. We will conclude the paper with a summary of our research.

II. BACKGROUND

The JPL PBM software system grew out of the NASA Deep Space Network program. The goal is to provide automated mission and service infrastructure management based on high level mission requirements. Figure 1 depicts the high level system architecture. The Mission Management Engine receives mission plans and uses a domain knowledge-base to enable the autonomous understanding and control of mission execution based on mission requirements and goals. The Mission Control Engine (MCE) and the Service Management Engine (SME) provide the real-time mission and service managements. The MCE and SME also disseminate executable policies to distributed policy-enabled agents to control applications and infrastructure.

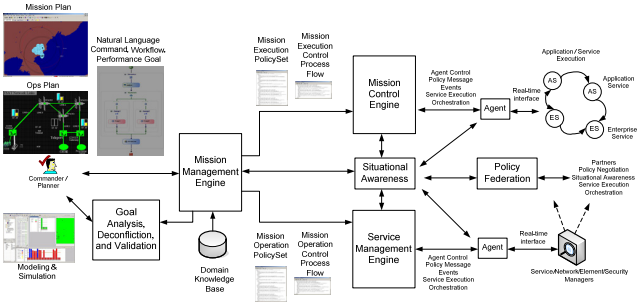

Figure 1. JPL PBM System High Level Architecture

The software implementation of the high level PBM system architecture is shown in Figure 2. The key software components are SHINE, HUNTER, SAFE, Frontier, and Policy-Enabled Agent.

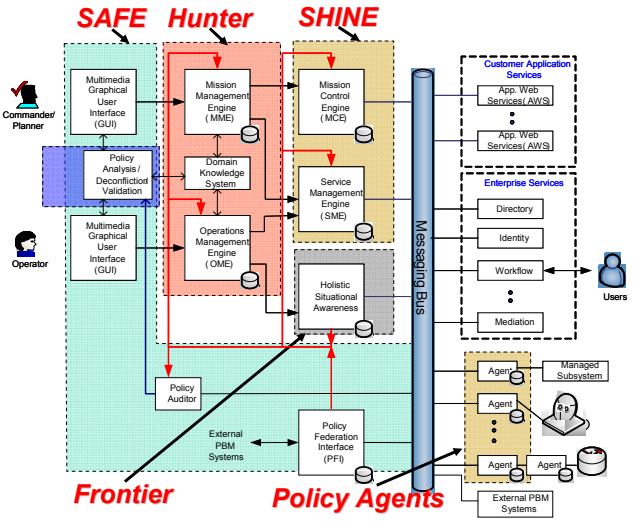

Figure 2. JPL PBM System Software Components

SHINE

The Spacecraft Health INference Engine (SHINE) is a rule based inference engine. It is the intelligent reaction component of our system that monitors for network events and, when observed, performs one or more actions. We take an inference-based rule approach for representing our network policy because it significantly enhances the users' ability of embedding advanced situational awareness and understanding within their policies. We use SHINE because it is a high performance inference engine that compiles network policies into highly efficient C++ code resulting in inference speed on the order of millions of rules per second. This provides the critically needed real-time performance that is required to monitor and react to network events. The SHINE rule engine has been benchmarked to execute greater than 200,000,000 rules per second on a standard 3.2 GHz Windows XP desktop PC and more than 33,000,000 rules per second on embedded hardware. SHINE has one of the smallest execution memory footprint compare to other rule engines. It is a proven technology used in many applications and products.

FRONTIER

A PBM system is only as good as the data from which it operates, and FRONTIER is the eyes and ears of the network. It provides SHINE with the information that it needs to make its decisions. FRONTIER works by monitoring all of the required software and hardware data sources of the network and transforms the large quantities of real-time data into information knowledge represented as SHINE rules. Essentially it provides situational awareness for the reasoning components.

HUNTER

Hunter is a natural language understanding system used to generate PBM policies from conversational language descriptions. It provides the capability for the automatic generation of PBM policies from high-level specifications whose interpretation is based on intent rather than the way in which the policy is being stated. Hunter can be used in combination with existing PBM systems to provide an integrated solution for the control and management of a complex network infrastructure. Hunter fills an important gap in that it provides a high-level interface to a complex system without the need of learning a special purpose language.

SAFE and Policy Editor Tools

The Secured Advanced Federated Environment (SAFE) Project [7] utilizes policies to dynamically bring up, reconfigure, and tear down network and security environments called Micro Security Domains (MSD). The goal of SAFE was to provide secured communication channels among all of the NASA centers and their affiliates. At the pinnacle of the project, six NASA centers, including JPL, incorporated SAFE technology to allow projects to transparently and safely share information between project partners at different NASA centers. The SAFE technology uses high level policies to manage lower level network-based

and security-based protocols and systems.

The Policy/Process Editor tool of the JPL PBM software evolved from the SAFE Project. This tool allowed operators as well as end users the ability to graphically enter policies as nested rules, manage federated policies, force inheritance of organization level policies, and use HUNTER to allow complex natural language policies as inputs to the tool. This provides a new method of organizing policies based on business process contexts. Policies and rules can now be combined into a workflow to establish visual relations between policies and procedures. It allows the user to build an extendable library of templates, giving institutions the ability and option to re-use previously established policies, rules, procedures, logic, and knowledge in any new workflows they use the editor to generate.

Policy-Enabled Agent

The SHINE rule engine is used to implement the Mission Control Engine (MCE), Service Management Engine (SME), and Policy-Enabled Agents (PEA). The MCE, SME, and PEA are at the core of our PBM system. The JPL PBM system involves not only the merge/negotiation of different network and security management policies, but also the dissemination of new policies to geographically distributed networked devices that enforce them. The MCE, SME, and PEAs realize these functionalities.

The MCE/SME act as the central policy coordinator for all PEAs. Since our task for the Test & Evaluation (T&E) environment focuses on infrastructure management, our discussion within this paper will be on the SME. We will also refer the SME as the policy server as opposed to the PEA. In the T&E environment, the policy server runs on computers within the test command center. The PEAs, located within test range networks, respond to commands and policies from the policy server by interpreting them to effect corresponding actions on their respective hosts and local test range networks. Since a typical test network comprises of a centralized test command center and many geographically-distributed test range networks, there are at lease as many PEAs as test range networks. Through the SAFE Policy Editor's federated policy management capability, a policy server can also control PEAs in different management domains through federation with the policy server in those domains. In a typical operational scenario, the policy server receives new policies from the user interface (Policy/Process Editor), transforms them into appropriate commands/policies/executable-code for the PEAs, distributes them to all PEAs, commands tests to be run, and collects results from distributed PEAs. The PEAs correspondingly participate in all activities as dictated by the policy server, as well as, provide regular updates to the policy server regarding their operational status.

III. SYSTEM ARCHITECTURE

The goal of the PAM Project is to develop the next generation test infrastructure management technology to enable automated netcentric systems testing across the DoD test infrastructure. In order to achieve this goal, the PAM technology needs to enable autonomic management of a test infrastructure spanning geographically distributed test ranges belonging to different organizations. Because of security reasons, end-to-end self-management of infrastructure belonging to other organizations is generally not acceptable. This is further complicated by the requirement to provide dynamic management to meet the challenges of future netcentric systems. This requires dynamic formation of autonomic network federations in seconds with very limited human involvement. Because different organizations have different goals, the dynamic federations need to operate within the constraints of dynamically changing groups of goals.

The JPL PBM software system provides the foundation for automated infrastructure management between federated partners. However, in order to achieve dynamic management capability, a number of additional capabilities must be added:

(1) Computer readable infrastructure description: In order to dynamically manage infrastructure in other organizations, the PBM software system must be able to understand the end-to-end infrastructure. Available network infrastructure information varies widely between organizations. It can range from a configuration management database in a network management system to a Visio drawing. Because of technology changes and differences in organization's implementations, it is difficult to have a uniform way of representing application and network infrastructure without constantly updating the standard. Furthermore, the representation must accommodate the privacy requirements of different organizations so that sensitive parts of the infrastructure can be hidden.

(2) Real-time policy negotiation: In order to satisfy the constraints of the dynamic group of organizations, a mechanism to achieve peer-to-peer policy negotiation in real-time must be developed. Today's client-server service negotiation [8] and peer-to-peer trust negotiation [9] are insufficient to accomplish the dynamic policy negotiation needs.

(3) Capture and reuse domain knowledge: Besides policy constraints, an organization often has a specific business process that is different from other organizations. A federation also needs to ensure their business process is acceptable to federated partners. Some operations may involve combinations of business processes. A mechanism to capture and dynamically reuse this domain specific knowledge is necessary.

(4) High speed reasoning component: A high-speed reasoning component is needed to ensure mission goals are accomplished. This reasoning component must work in both the command center and distributed PEAs. The reasoning component enables the PEAs to satisfy mission goals through distributed control of network components.

A. Solution Components

The following details our solutions to these additional

capabilities.

Integration Markup Language

The Integration Markup Language (IML) is designed to address the problem of dynamically exchanging infrastructure descriptions across organizations. It is based on the Semantic Web technology so that computer software can read and understand network and application configurations. The basic problem is the difficulty of enforcing uniform infrastructure description standards across organizations. The idea behind IML is to use a base standard schema where each company can extend by publishing a computer readable extended schema.

We developed the IML based on the Open Grid Forum NML [10] and the NDL [11]. The intent of the NML is to describe optical hybrid network topologies. NML is too simple for our purposes. We extended the NML schema with a schema derived from the DMTF CIM. We also extended the NDL to cover application configuration descriptions. Figure 3 shows a small sample schema from the IML. Figure 4 shows some sample portions from an IML network description file for an example network.

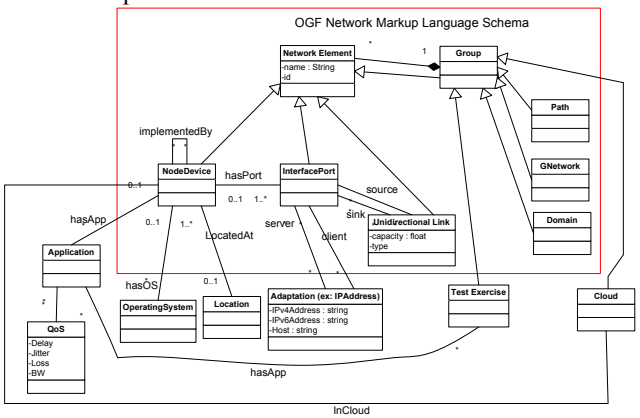

Figure 3: Example IML Schema.

```
<?xml version="1.0" encoding="UTF-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:iml="http://pbm.jpl.nasa.gov/iml#">
.....
<iml:OperatingSystem rdf:about="#Windows">
<iml:name>Windows</iml:name>
  </iml:OperatingSystem>
.....
<iml:Recipe rdf:about="#TestApplication1">
<iml:name>TestApplication1</iml:name>
  </iml:Recipe>
.....
<iml:NodeDevice rdf:about="#PAXRIVER_Host1">
<iml:name>PAXRIVER_Host1</iml:name>
<iml:locatedAt rdf:resource="#PaxRiver.navy.mil" />
<iml:hasPort rdf:resource="#PAXRIVER_Host1:eth1" />
<iml:hasOS rdf:resource="#Linux" />
<iml:hasApp rdf:resource="#TestApplication1" />
  </iml:NodeDevice>
.....
<iml:UnidirectionalLink rdf:about="#PAXRIVER_router1.PaxRiver.navy.mil:2">
<iml:name>PAXRIVER_router1.PaxRiver.navy.mil:2</iml:name>
<iml:source rdf:resource="#PAXRIVER_router1.PaxRiver.navy.mil:2" />
<iml:sink rdf:resource="#PAXRIVER_Host1:eth1" />
  </iml:UnidirectionalLink >
```

Figure 4: Sample Portions of an IML File

Based on the published IML schema, an organization can convert the network and application configurations into IML descriptions. The IML file can be accessible by federated partners. The policy server and PEAs can then read the IML file and understand the network and application configurations of the partner organization. The flexible structure of IML allows an organization to describe the network down to the device level or the network can be described as a cloud with ingress and egress points. This allows an organization to hide restricted information based on the level of trust of the partner organizations.

Policy Negotiation Algorithm and Protocol

To achieve a technology which realizes the PAM's goal for a federated, automated, reliable, and fast network management, automating the negotiation of the network management policies of different organizations is of particular importance.

Policy negotiation in this context means the process of determining the "best" communication policy that all of the parties involved can agree on. The core problem here is how to reconcile the various (and possibly conflicting) network management protocols used by different organizations. The solution must use protocols available to all the parties involved, and should attempt to do so in the best way possible. Which protocols are commonly available, and what the definition of "best" is will depend on the parties involved and their individual management priorities. Large-scale web security systems usually involve cooperation between domains with non-identical policies. Developing practical, sound, and automated ways to compose policies that bridge these differences is a long-standing problem. One key subtlety is the need to deal with inconsistencies and defaults where one organization proposes a rule on a particular feature and another has a different rule or expresses no rule.

A desirable policy negotiation methodology should have the following characteristics. First, the method must be simple and intuitive to policy designers. Second, the method should have a formal foundation that allows careful reasoning about the correctness of algorithms and consequences of composition in boundary cases that could be exploited by attackers. Third, enforcement should be efficiently implementable on existing programming platforms and consistent with existing or proposed standards.

In the general case, policy negotiation is an intractable problem [12, 13]. But this fact does not rule out the existence of efficient methods for specific classes of policies, especially types of policies implemented in specifically desired applications. As a result, efficient policy negotiation methods have been suggested for some classes of policies. For example, a promising method is suggested in [14] wherein policies are represented in defeasible logic and composition is based on rules for non-monotonic inference. In this system, policy writers construct meta-policies describing both the policy that they wish to enforce and annotations describing

their composition preferences. These annotations can indicate whether certain policy assertions are required by the policy writer or, if not, under what circumstances the policy writer is willing to compromise to allow other assertions to take precedence. Meta-policies are specified in defeasible logic, a computationally efficient non-monotonic logic developed to model human reasoning.

One drawback of the method described in [14] is that at one point, the algorithm starts an exhaustive search in the set of all subsets of the set C of conclusions of the defeasible logic defined by the policies at hand. Although the propositional defeasible logic has linear complexity [15], the set C here may be of large size, especially for the real-life practical cases. This phenomenon leads to an inefficient, exponential explosion complexity.

The JPL team has developed a new algorithm for the policy combination that overcomes this deficiency [16]. This new algorithm substitutes the search in the set of all subsets of C by a linear search through the elements of C. Our new algorithm has an overall quadratic time complexity with respect to the size of the initial policies. Based on this new algorithm, the JPL team has developed the Defeasible Policy Composition (DPC) tool (coded in C++) which we describe briefly.

The solution of [12] to policy negotiation, and subsequently the solution used by the new JPL engine, uses "defeasible" logic to describe communication policy constraints and priorities. Defeasible logic is non-monotonic, and contains three different types of rules:

- Strict Rules: strict "if–then" statements in the classical sense;
- Defeasible Rules: "if this, then usually that" statements that can be defeated by contrary evidence;
- Defeater Rules: contradict the outcomes of defeasible rules.

Defeasible logic also has a sense of priorities, so that one rule can have priority over another. If they are contradictory, the superior rule overrides the inferior one. This form of logic provides a more expressive power than the classical logic. For example: "Birds can usually fly, but penguins cannot" can be expressed as a defeasible rule for birds ("if it's a bird, then usually it can fly"), and a defeater rule for penguins ("if it's a penguin, it can't fly"). Of course, the same statement can be expressed by regular strict "if–then" as "if it's a bird and it's not a penguin, then it can fly." Both express the same thing, but defeasible logic expresses it in a more natural manner. Other merits of defeasible logic can be found in [17].

The policy negotiation tool, DPC, reads in two files specifying the policies that we want to combine, and outputs a single file describing the means of communication which satisfy both input policies, if any can be found. For the sake of usability and portability, the file format used is an XML-based language called RuleML [18]. Then the XML data from the input files are parsed into tree objects: facts, rules, and priority relations. Strict rules are treated as Requirements to be fulfilled. All other rules (i.e., defeasible and defeater rules) are treated as Reasoning rules. Using techniques described in [19], the Reasoning rules are transformed to an equivalent set R with no defeater or priority relations. By utilizing the inference methodology of [15], the set C of all logical conclusions of the defeasible theory R are derived. Then the tool implements our new algorithm [16] to find a subset P of C which is consistent with the Requirements. The statements of P (if it is not empty) define the combined policy which satisfies the requirements from both input policies, and it will be produced as an XML file.

By iterating the above procedure for combining two policies, we have designed and tested a method for combining multiple policies.

For benchmarking, we used random inputs of policies of combined sizes from 100 to 4000 rules and applied the C++ DPC tool and record the performance time in seconds. The horizontal axis in Figure 5 denotes the total number of rules and the vertical axis denotes the performance time in seconds. For most practical PBM systems with less than 500 combined rules, policy negotiation can be completed in less than a second.

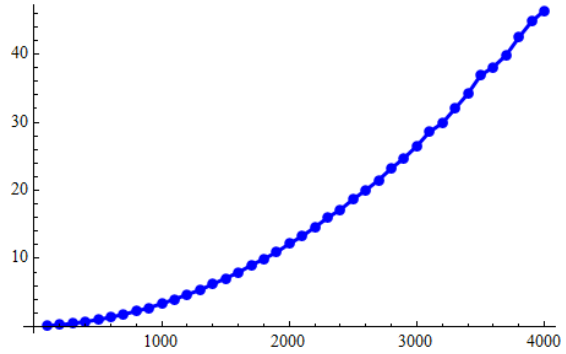

Figure 5: Policy Negotiation Tool Performance Benchmark

Knowledge Capture and Execution

One of the problems in end-to-end network management is that an organization often only allows a specific set of business processes. A key point is to produce a system that understands the acceptable business logic of the organizations involved. We need a way to capture business process as domain knowledge and automatically select and execute the acceptable knowledge. We achieve this through a number of different ways. First, we developed a sophisticated knowledge capture tool that uses a combination of graphical programming techniques enhanced with natural language input. The user uses these capabilities to build a toolbox of techniques to autonomously and safely correct a variety of network problems. These solutions are organized into a hierarchically organized library, which can be selected by the reasoning software to become a component of more complex network remediation algorithms.

The user interface for capturing domain knowledge is shown in Figure 6. The tool is based on the Microsoft Workflow Foundation. We enhanced the software with the

capability to enter policies at each activity step. This enables suitable knowledge to be chosen to solve the right problems. The output of the knowledge is converted into the SHIELD language so that the process can be executed by a SHINE engine. SHIELD is one of the input languages for SHINE and is an XML-based language for representing procedures and rules.

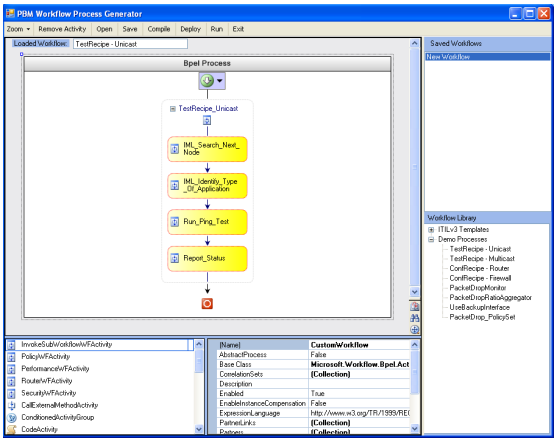

Figure 6: User Interface to Capture Domain Knowledge

Reasoning Component (Guardian)

We are using an innovative approach that we call Ontology-based Reasoning Semantic Web (RSW). The RSW provides the core reasoning infrastructure utilizing a tightly coupled set of cooperating agents, each being an expert in one or more specialized areas, e.g., mission requirements, language understanding, policy constraints, test procedure generation, etc. This methodology was chosen because it is not possible to have just one monolithic reasoner that operates in isolation. The process of understanding requires the integration of many different sources of knowledge and data in order to produce a solution.

Guardian works as follows. After a test mission is defined, the test requirements are translated to the IML format. This is passed to the Guardian reasoner that analyzes the IML for all subjects and actions for which models have been defined in the RSW. These models are hieratically defined so models deeper in the hierarchy inherit non-contradictory information from their parent models. Additionally, models can have collateral inheritance that provides the capability of reasoning and abstraction from incongruent logical domains. During reasoning, logical consistency is guaranteed by a truth maintenance agent that is layered on top of the RSW. Each subject and action (object) can have one or more models associated with it. A model describes the characteristics of the object, tests that must be performed with respect to certain actions or other subjects, etc. Associated with an action is a recipe that defines the action's function when combined with other actions. This provides the capability to have subject-focused reasoning so actions can be used in a natural way. For example, assuming we experienced "UAV video transmission failure" in a test exercise. Because UAV is a subclass of the "Surveillance Aircraft" and "Surveillance Aircraft" is a subclass of the "Aircraft", Guardian will apply the "Aircraft Video Testing Procedure" recipe in the knowledge base to identify the problem with the failure.

Using the RSW approach and the domain knowledge base from federated partners, the Guardian can select the acceptable processes (recipes). The output from Guardian is a set of instantiated recipes, utilizing the SHIELD interpreter outputs as functional directives. The recipe is then compiled by SHINE and disseminated to the distributed PEAs for execution. Only then can the PEAs perform monitoring and corrective activities and report situational awareness data back to policy server and Guardian for further response and reasoning.

B. *End-to-End System Operation*

Using the tools from the JPL PBM system coupled with Guardian we developed a system that enables dynamic, federated, autonomic networking within the T&E environment. Figure 7 depicts this software system architecture. The Guardian software receives the mission requirements and tries to satisfy the mission goals. If the mission requires a formation of federation, Guardian will command the policy server to federate with the policy servers in other domains. This may involve the exchange of allowable network and application configuration IML files, access control policy negotiations, network protocol or waveform policy negotiations, allowable testing tool/protocol negotiations, etc. During the federated mission operations, if a mission requirement can not be met, Guardian will analyze the network IML file to identify the potential problems. It will then apply the suitable domain knowledge and command the policy servers and PEAs to perform automated corrective sequences. The PEAs will use the IML files to perform automated step-by-step measurement and testing until the problem is identified. Depending on the knowledge base, the measurement and testing can range from simple connectivity tests, communication protocol testing, and to sophisticated end-to-end deep packet inspection for "goodput" measurements. Based on the problems that were identified, the PEAs can inform the operators to fix the problems or, if allowed by policy, the PEAs will automatically perform the corrective actions.

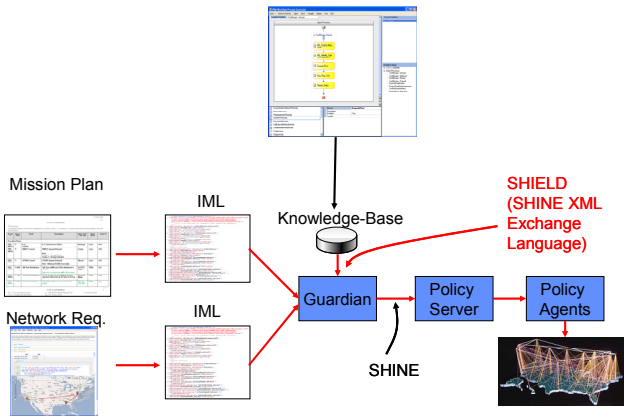

Figure 7: End-to-end Software Architecture

IV. SYSTEM IMPLEMENTATION

PBM tool suite consists of several software applications, which includes: (1) Policy/Process Editor user interface applications, (2) Policy Server, (3) Guardian, and (4) Policy Agents. The tool suite runs on Windows-based PCs but we are in the process of building deployments for Linux as well.

The communication protocol between Policy Server and Policy Agents can be changed depending on the test network setup. We have developed software for the communication layer using the TCP protocol, the JPL SharedNet tactical message bus, as well as, the Test and Training Enabling Architecture (TENA) protocol, which is a publish-subscribe network protocol on an underlying CORBA structure.

A test network setup (Figure 8) involves a central Test Control Center (TCC) and several geographically distributed test ranges. The TCC will have the Policy/Process Editor user-interface applications, Guardian, and the Policy Server application on its local computers. Each range network will have one Policy Agent application per test computer. The Policy Agent application is installed on a local computer as a "Trusted Application" by the range network administrator giving the Policy Agent super-user privileges on that host as well as other network resource access privileges. The range network administrators can also impose network policies that the Policy Agent must adhere to during its operation over the local range network. This aspect of our PBM technology is implemented by a multi-party policy negotiation phase, which is initiated by the Policy Server during the test federation formation phase. This is performed before any distributed testing is done on the test network to select the set of common tools that the different ranges and test command centers can use for performing their test exercises. This multi-party policy negotiation is done each time a test exercise runs to determine what common tools are used by all parties involved in the test exercise. If this automatic multi-party policy negotiation phase fails to qualify the test exercise, then the Policy Server will provide situational awareness to the operators.

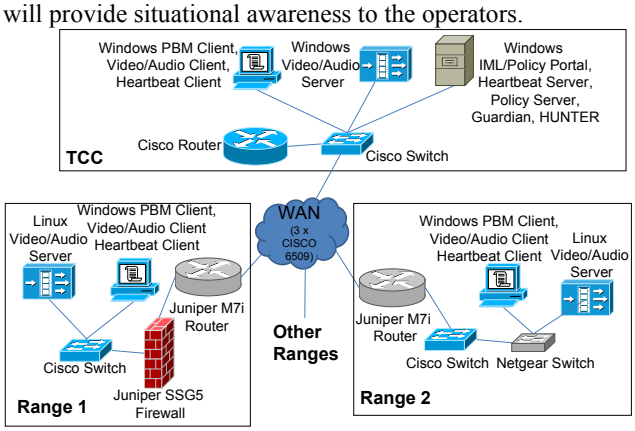

Figure 8: Testbed Network Diagram

The workflow of the system is described as follows:

(1) The federated test ranges in a joint test exercise is identified.

(2) The Policy Server distributes the network IML files to the Policy Agents running in different test ranges. This causes the Policy Server and all the Policy Agents to be aware of the available network resources (hosts, switches, routers, firewalls, etc.) that are participating in the planned test operation.

(3) A test event IML file is generated. This file contains information about the operational details of the planned test on the network.

(4) The Policy Server distributes the test event IML file to the Policy Agents. This causes the Policy Server and the Policy Agents to have knowledge about different kinds of tools and programs that can run during the test exercise.

(5) The Policy Server initiates a multi-party policy negotiation process to ensure all the Policy Agents can be set up to use a common set of tools and procedure for the test operation.

(6) After the automatic multi-party policy negotiation process is successful, Guardian is ready to build any custom test procedures that are required during the test operation. This is accomplished by:

a. Selecting the most appropriate tools to use based on multi-party policy negotiation outputs;
b. Generating the custom test procedures leveraging the knowledge base;
c. Transforming these high-level test procedures to C/C++ code using SHINE rule-based processing subsystem;
d. Building the binary executable code for the target operating systems. These binaries are then distributed to the Policy Agents enabling them to handle the test operational steps successfully;

(7) The final step in the workflow is to start the distributed test process:

a. The Policy Server will send the "start" command to each of the Policy Agents and each Policy Agent will conduct all the operational steps for the planned test and report the result to the Policy Server.
b. Guardian will analyze the results and take any necessary corrective actions by issuing new policies and commands to the distributed PEAs.
c. If a PEA detects a problematic situation during its test steps, and it has knowledge of a corrective procedure, it can either perform automated corrective procedure or, if the policy does not allow automated actions, report the problem and corrective action to a local, range-operated Situational Awareness Console. The operator involved capability is necessary for the range administrator to understand the complete operational context and agree to appropriate corrective actions. This is a very important part of our PBM system because it provides the option to ensure that the actions of the "Trusted Application" are completely visible to the local range administrator, thereby ensuring compliance with local security policies.

V. USE CASES

We will discuss two use cases in this section. The use cases

show that our system can autonomously detect, diagnose, and correct network misconfigurations and failures.

Use Case #1

The first use case involved a firewall misconfiguration issue that isolated the TR1 workstation from joining the TCC range's multi-cast stream. This scenario was taken from a real test exercise involving test ranges around the country. The error took dozens of operators over 6 hours to identify and fix, while our solution accomplished the same task in seconds.

To begin, several setup steps had to take place. First, the operator inputs a set of policies dictating which testing tools can be utilized within each range. For our demonstration, *ping* and *traceroute* were chosen as possible network connectivity tools and *iperf* and *mcfirst* were chosen as possible multi-cast configuration tools. This set of policies is submitted through a web-based policy terminal. Second, the operator accesses the IML web portal to input two IML files, the Network Description IML and the Test Event IML. The Network Description IML details the network topology of the testbed with identities and descriptions for end nodes, network nodes, and network links. The Test Event IML details the applications and target platforms, which include multi-cast and other end nodes, of which the operator determines which will participate in the test event. Once received, the Policy Server will automatically disseminate the contents of these two files to all of the agents in the testbed. Each agent will a) utilize the Network Description IML to determine its own identity, b) match that identity with the Test Event IML to extract the tools and applications available to itself, and c) reference both IML documents to determine the target test server and the shortest path from itself to that target end node. The agents now perform the pre-negotiation step by correlating applications available on the workstation with the range policy on which applications are allowed to run. The result is a workstation policy XML file containing a non-unique set of testing tools.

The third setup step is for the operator to utilize the Process Editor to input the procedure for performing Network Connectivity Tests (NCT) and System Configuration Tests (SCT) with the available testing tools. Once deployed, the Process editor translates the Business Process Execution Language (BPEL) XML procedures into SHIne Execution Language (SHIELD) and sends off the SHIELD XML files to the SHIELD interpreter. This interpreter translates the XML into C-target compilable code and sends that code to the Policy Server. Upon receiving the NCT and SCT source code, the Policy Server initiates the three-party negotiation mechanism. First, the Policy Server queries for the local workstation policy and disseminates that policy to the workstations at the other two test ranges. Second, each workstation in TR1 and TR2 performs one level of negotiation between its local workstation policy and the TCC workstation policy files using the Defeasible Policy Combination (DPC) tool. The result is a common set of tools allowed and available between a) TR1 workstation and TCC workstation and b) TR2 workstation and TCC workstation. Third, these two new workstation policies are returned to the Policy Server where the second level of negotiations is conducted using the DPC tool. The final result is one Policy Set of a unique selection of one network testing tool and one system configuration testing tool that is common to, allowable, and available for all three test ranges. This final policy XML is disseminated to all three workstation agents. Finally, the operator uses the Process or Policy Editor to enter in a policy stating that network and system configuration tests should be run one hour prior to the scheduled test event. This policy is submitted and accepted by the Policy Server, and the rest of the scenario becomes almost completely automated.

On to the actual scenario, one hour prior to the scheduled test event, the Policy Server invokes the network and system configuration test policy. The policy server sends a start command to each workstation involved in the test event, causing each agent to start its local multi-cast server. After this point, the TR1 and TR2 multi-cast server start broadcasting locally while the TCC's multi-cast server performs a global broadcast of its multi-cast stream. To confirm end-to-end connectivity in preparation for the test event, each workstation only needs to be able to join the stream published by the TCC's multi-cast. TCC's workstation as well as TR2's workstation join the stream just fine and can conclude that they are ready for the test event. However, TR1's workstation cannot join the stream due to the misconfiguration of the TR2 firewall. After a few attempts, the TR1 workstation agent auto-detects that there is a problem to be solved and begins to invoke its NCT procedure as part of the auto-diagnose process. The agent looks in the final Policy Set for the network connectivity testing tool and uses it to contact the TCC multi-cast server. This query returns a successful acknowledgement, telling the agent that network connectivity is not the problem. Next, the TR1 workstation agent will invoke the SCT procedure by sending out a broadcast query for multi-cast servers. This time, TR1 receives a multi-cast acknowledgment from the TR1 multi-cast server but still nothing from the TCC multi-cast server. This tells the agent that the multi-cast application is not the problem, and since the other two workstations can connect with no problems, the agent can reasonably deduce that the firewall is causing the problem. The agent can then connect into the firewall, and detects that the multi-cast configuration is pointing to a non-existent host on the network. Finally, the agent presents the TR1 operator with the agent's findings and a potential solution. If the agent is allowed to perform automated fixes, the agent sends a configuration update to the firewall to correct the host's identity. Once again, the agent tries to connect into the multi-cast stream. This time, the connection is successful, the agent logs the findings and success, and goes to sleep for the next time it is called.

Use Case #2

The second use case involved the TR2 primary link failure that effectively isolating the entire test range from the

remaining network environment. The primary link breakage scenario builds on the policies and procedures used in the firewall misconfiguration scenario. There is only one extra setup step which is to submit a set of policies that would call on the PBM system to auto-correct primary link failures at managed switches. After these policies are submitted, the actual scenario starts.

Here is the second case of required operator involvement. The TR2 operator simulates a primary link failure at the managed switch in TR1 by yanking out the primary Ethernet link cable from the switch. By doing this, TR2 is effectively isolated from the rest of the network. Because the TR1 and TR2 agents have been sending regular heartbeat messages to the Policy Server, the server auto-detects that a connectivity failure has occurred with the TR2 workstation. The Policy Server then invokes the NCT procedure as part of the auto-diagnosis process and determines that the managed switch is the furthest node that can be reached. Then, from the Network Description IML, the Policy Server determines the existence of the backup link on the managed switch and activates it as part of the auto-correction process. Several seconds later, the TR2 range comes back online with the continuation of the heartbeat messages from the TR2 agent.

Beside the two use cases presented, we believe our technology has the potential to be applied to a wide range of federated autonomic networking problems. In the next phase of the PAM project, we will use this approach to address critical security management problems for the T&E environment. In the future, we would like to apply our technology to other critical problems such as automated collaborative cyber operations, cloud computing, and interoperable smart grid network management.

V. SUMMARY

We have described our approach to federated autonomic networking for the test and evaluation environment. Our approach allows dynamic formation of federated partners through IML information exchange and policy negotiation process. Our knowledge capture and execution process enables the selection and reuse of appropriate business processes that can be automatically executed. Our federated policy server and PEA architecture enables automated end-to-end measurement, testing, and corrective actions. The system is designed to operate in both trusted and untrusted modes so corrective actions are not automatically performed by the system, unless instructed to do so. This means the system can be applied in critical operations because the operator is never out of the loop unless they want to be.

By introducing the federated autonomic networking capability into the T&E environment, we open the possibility for the next generation of netcentric system testing where complex multi-range testing can be set up in seconds, dynamic changes to test exercises can be accomplished in real-time, and multiple test exercises can be conducted in parallel. Also, this approach can be extended and generalized to enable dynamic, federated, and autonomic network management for the future Internet. The end result is significant reduction in test planning times and great cost savings in managing test exercises.

ACKNOWLEDGMENT

The PAM team appreciates the direction and support of Mr. Robert Heilman, Mr. Gilbert Torres, Mr. Jim Buscemi, Mr. Kent Pickett, and Mr. Ryan Norman. The PAM team also wants to express deep appreciation to Mr. Jim Ledin and the InterTEC team for their support on defining the critical T&E problems. We would also like to thank Mike Glazer for his contribution for the IML Database Software development.

REFERENCES

[1] C. Tschudin and C. Jelger, "An "Autonomic Network Architecture" Research Project," *Proceeding PIK,* vol. 30, no. 1, pp. 26-31, Jan-Mar 2007.

[2] R. Chaparadza, "Requirements for a Generic Autonomic Network Architecture suitable for Standardizable Autonomic Behavior Specifications of Decision-Making-Elements for Diverse Networking Environments", International Engineering Consortium (IEC), Annual Review of Communications Volume 61, December 2008

[3] Ipanema Technologies, http://www.ipanematech.com/

[4] IBM Corporation, http://www-01.ibm.com/software/tivoli/autonomic/

[5] B. Jennings, R. Brennan, W. Donnelly, S. Foley, D. Lewis, D. O'Sullivan, J. Strassner, S. van der Meer, "Challenges for Federated, Autonomic Network Management in the Future Internet," 1st *IFIP/IEEE International Workshop on Mgm. of the Future Internet.*, Jun. 2009.

[6] P. Rubach and M. Sobolewski, "Dynamic SLA Negotiation in Autonomic Federated Environments," SpringerLink Lecture Notes in Computer Science, Vol. 5872, pp. 248-258, Nov. 2009.

[7] Chow, E., Chew Spence, M., Pell, B., Stewart, H., Korsmeyer, D, Liu, J., Chang, H., Viernes, C., Goforth, A., "Secured Advanced Federated Environment,", IEEE WETICE Conference, June 2003..

[8] V. Cheng, P. Hung, and D. Chiu, "Enabling Web Services Policy Negotiation with Privacy preserved using XACML," Proceedings of the 40th Hawaii International Conference on System Sciences, 2007.

[9] J. Li, D. Zhang, J. Huai, J. Xu, "Context-aware trust negotiation in peer-to-peer service collaborations," SpringerLink Peer-to-Peer Networking and Applications, Vol. 2, No. 2, pp. 164-177, Mar. 2009.

[10] Open Grid Forum, NML-WG, https://forge.gridforum.org/sf/wiki/do/viewPage/projects.nml-wg/wiki/Deliverable_2.

[11] J. van der Ham, F. Dijkstra, F. Travostino, H. Andree and C. de Laat. "Using RDF to Describe Networks", Future Generation Computer Systems, Feature topic iGrid 2005, October 2006.

[12] L. Gong and X. Qian, The complexity and computability of secure interoperation, *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 190–200, 1994.

[13] P. McDaniel and A. Prakash, Methods and limitations of security policy reconciliation, *ACM Transactions on Information and System Security*, vol. 9, no. 3, pp. 259–291, 2006.

[14] A. Lee, J. Boyer, L. Olson, and C. Gunter, Defeasible security policy composition for web services, *Proceedings of the fourth ACM workshop on Formal Methods in Security*, pp. 45–54, 2006.

[15] M. J. Maher, Propositional defeasible logic has linear complexity, *Theory and Practice of Logic Prog.*, vol. 1, no. 6, pp. 691–711, 2001.

[16] F. Vatan and J. Harman, A new efficient web services policy combination, *JPL Internal Report*, August 2009.

[17] D. Nute, Defeasible logic, Handbook of Logic in Artificial Intelligence and Logic Programming, vol. 3, pp. 353–395, 1994.

[18] *The Rule Markup Initiative,* www.ruleml.org.

[19] G. Antoniou, D. Billington, G. Governatori, and M. J. Maher, Representation results for defeasible logic, *ACM Transactions on Computational Logic*, vol. 2, no. 2, pp. 255–287, 2001.