Dynamic Communication Resource Negotiations

Edward Chow
Farrokh Vatan
George Paloulian

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA USA

Steve Frisbie
Zuzana Srostlik

SPAWAR Systems Center Pacific
San Diego, CA USA

Vasilios Kalomiris
Daniel Apgar

Army CERDEC S&TCD
Aberdeen Proving Ground, MD

***Abstract*— [1]**

Today's advanced network management systems can automate many aspects of the tactical networking operations within a military domain. However, automation of joint and coalition tactical networking across multiple domains remains challenging. Due to potentially conflicting goals and priorities, human agreement is often required before implementation into the network operations. This is further complicated by incompatible network management systems and security policies, rendering it difficult to implement automatic network management, thus requiring manual human intervention to the communication protocols used at various network routers and endpoints. This process of manual human intervention is tedious, error-prone, and slow. In order to facilitate a better solution, we are pursuing a technology which makes network management automated, reliable, and fast. Automating the negotiation of the common network communication parameters between different parties is the subject of this paper. We present the technology that enables inter-force dynamic communication resource negotiations to enable ad-hoc inter-operation in the field between force domains, without pre-planning. It also will enable a dynamic response to changing conditions within the area of operations. Our solution enables the rapid blending of intra-domain policies so that the forces involved are able to inter-operate effectively without overwhelming each other's networks with in-appropriate or un-warranted traffic. It will evaluate the policy rules and configuration data for each of the domains, then generate a compatible inter-domain policy and configuration that will update the gateway systems between the two domains.

Keywords- Policy-Based Management; Policy Negotiation; Network Management

I. INTRODUCTION

A policy is formally defined as a collection of rules, where each rule consists of a set of conditions and a set of actions. The conditions define when the policy rule is activated. Once a policy rule is activated, actions embedded in that rule may be executed or must be executed, depending on the characteristic of that rule.

For managing large-scale complex systems that dynamically change their state to adapt to changes in the application requirements, one of the promising technologies is policy-based network management. This approach allows dynamic modification of the policy rules without need for human operators' intervention. Such policy-based technology allows automatic management of large systems and frees the manager from monitoring the equipment and systems directly and provides a systematic method for producing and modifying policy rules.

The technology described in this paper is devoted to providing a solution for one of the main challenges in network management and communication. Since many of the tools used by the various divisions do not have the ability to communicate network management data with each other, automatic network management is very difficult to implement and manual human intervention to the communication protocols used at various network routers and endpoints is required. This process of manual human intervention is tedious, error-prone, and slow. In order to facilitate a better solution, we are pursuing a technology which makes network management automated, reliable, and fast. Our focus is to address the challenge of automating the negotiation of the common network communication parameters between different parties when they wish to communicate.

Policy negotiation is the process of determining the "best" communication protocol that satisfies all requirements of all parties involved. The main challenge here is how to reconcile the various (and possibly conflicting) communications protocols used by different parties. The solution must satisfy the requirements of all parties involved, and should achieve it in an efficient way. Which protocols are commonly available, and what the definition of "best" is will be dependent on the parties involved and their individual communications priorities. Therefore, we are looking for a solution that on one hand should be simple and intuitive to the operator, and on the other hand should be implemented efficiently with minimal demand on existing operational computing equipment while complying with current and emerging standards.

[1] Portions of 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 Office of the Secretary of Defense Network Communication Capability Program, under NASA prime contract NAS7-03001, Task Plan Number 81-103508. Copyright 2012. All rights reserved.

For example, the Joint Tactical Radio System (JTRS) consists of several types of tactical radios, from hand-held radios to vehicle and aircraft mounted systems. When multiple forces inter-connect, a Service Level Agreement (SLA) must be established between these forces. This SLA typically addresses traffic shaping policies such as capacity for the other force's use, when their organic reach back is available and when it is not. An SLA also addresses traffic priority schemes, quality of services, and access control lists. The traditional method to establish an SLA is to negotiate the operational aspects of the agreement, followed by developing and testing the actual system configurations. Any problems discovered result in a re-negotiation of the operational aspects and another cycle of development and testing. Historically, this has been a time-consuming process that takes months or even years to complete.

In the emerging battle field, disparate forces are expected to be assembled and operate together in the field at a pace that does not allow for a lengthy SLA development phase. There is a clear need to dynamically negotiate technical parameters for a SLA based on pre-established operational policies for the two or more forces involved.

As for the policy negotiation problem in general, it is known that this is an intractable problem (technically, "NP-complete") [1], [2]. But this fact does not rule out the existence of efficient methods for specific classes of policies, especially types of policies implemented in specific desired applications. Efficient policy negotiation methods have been suggested for some classes of policies. One method that our approach is based on is a promising method suggested in [3] 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 relations describing their composition preferences. These relations can indicate the required rules, the conditions for compromising the rules, and the precedence relation among rules.

II. APPROACH

A. General Approach

We first describe a general mechanism of negotiation. In the beginning, each party receives a description and the subject matter of the negotiation. Based on this information, the party chooses an appropriate template for negotiation; and based on this template, a suitable negotiation policy strategy will be activated. Here the basic assumption is that each party is equipped in advance with a repertoire of policy negotiation strategies that can handle conceivable situations. Once a negotiation strategy is picked the negotiation process will start. Each round of negotiation begins with each party *offering* their requirements and what they are willing to make available to the other party. Except for the initial offer, each offer is obtained from the logic encapsulated in the negotiation strategy and the offers from the other parties. Once all parties provide their offers, a test will be performed to see whether these offers satisfy the requirements. Later we describe our mechanism for this test. If this test fails, then there are tests to

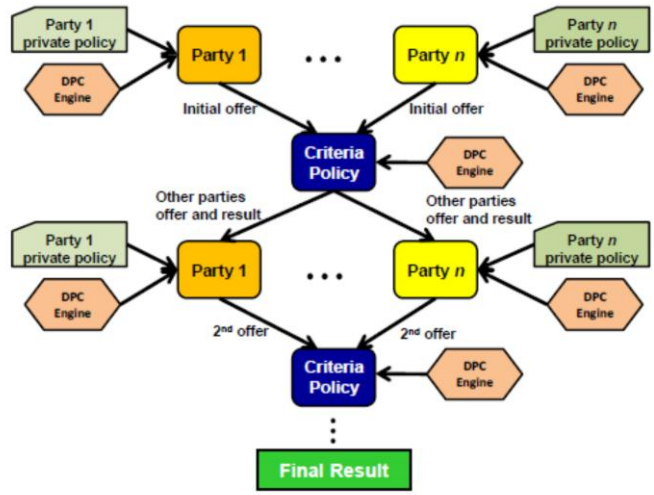

Fig. 1. Policy negotiation implementation.

see whether the negotiation process should continue; these tests may consist of some predetermined criteria or specifically a bound on the number of rounds. Also in the case that the offers do not satisfy the requirements, it is possible to introduce some criteria for modifying the negotiation strategies to avoid negotiation deadlock. To modify the negotiation strategy, help from the strategy repository or operator is possible.

To implement the above general scheme we have developed a concrete implementation. Fig. 1 shows the flowchart of this implementation. Each party has its own ***private policy*** that describes its requirements, preferences, and strategy for providing offers and revising them. In our implementation this private policy is formulated in *defeasible logic* (see examples in Section III). In our implementation the offers are (defeasible) logical conclusions of the union of the private policy, the results of the previous round, and the offers of others parties. Of course, the initial offer is the conclusion of the private policy only. To find the conclusions of defeasible logic theory, we utilize JPL's efficient defeasible engine, DPC (Defeasible Policy Combination) [5], [6].

In our implementation, the role of "referee" for deciding whether offers from parties satisfy the predefined requirements is played by ***criteria policy***. Like private policy, criteria policy is also formulated as a defeasible theory. Once the offers O_1, O_2, …, O_n, from the parties is received, the result is the (defeasible) logical conclusion of the defeasible theory obtained from the union of the criteria policy and O_1, O_2, …, O_n. Elements of the above policy negotiation implementation are summarized in Fig. 2.

We have also considered the case that negotiation can be performed in only one round. This method can be used in the cases that the policies of the parties allow so many alternatives that a subset of allowable choices provides a satisfying choice (see an example in Section III).

- ***Private Policy*** of a party: requirements, preferences, and strategy for providing offers and revising them
- Private Policy is formulated in ***defeasible logic***
- ***Offers***: logical conclusions of the union of the private policy, the result of the previous round, and the offers of other parties
- ***Initial Offer***: the conclusion of the private policy only
- ***Criteria Policy***: determines whether offers satisfy the requirements
- ***Result of a round of negotiation***: logical conclusion of the defeasible theory obtained from the union of the criteria policy and offers

Fig. 2. Elements of policy negotiation implementation.

B. Defeasible Logic

A defeasible theory [3], [4] has five different elements:

- facts,
- strict rules,
- defeasible rules,
- defeaters,
- superiority relations.

Facts are given or observed facts of a case which are presented by (logical) literals; i.e., a variable (or atomic formula) p or its negation $\sim p$. We also use this convenient notation: if q is a literal, then $\sim q$ denotes the complementary literal (i.e., if q is a positive literal p then $\sim q$ is $\sim p$; and if q is $\sim p$, then $\sim q$ is p).

Strict rules are rules in the classical sense; i.e., whenever the premises are true then so is the conclusion. For example,

$$penguin(X) \rightarrow \sim flies(X).$$

This rule means that is "if $penguin(X)$ is true then $flies(X)$ is not true" (or, in other words, "penguins don't fly").

Defeasible rule $A \Rightarrow p$, which means when all the literals in A are true then normally or typically p is true but can be defeated by contrary evidence. For example,

$$bird(X) \Rightarrow flies(X)$$

The meaning of this rule is that "if $bird(X)$ is true, then we may conclude that $flies(X)$ is true, unless there is other evidence, with higher priority, suggesting that it is not true" (or, in other non-technical words, "birds typically fly"). In the context of formulating policies, defeasible rules are used to express alternatives and possibilities.

Defeaters $A \rightsquigarrow p$ when all the literals in A are true one should not normally conclude that p is true. These rules cannot be used to draw any conclusions. Their only use is to block the conclusions of defeasible rules. In other words, they are used to defeat some defeasible rules by producing evidence to the contrary. For example, the rule

$$injured(X) \rightsquigarrow \sim flies(X)$$

will block a rule like $bird(X) \Rightarrow flies(X)$ since the knowledge that a bird is injured counteracts our intuition that birds usually fly. The main point is that the information that a bird is injured is not sufficient evidence to conclude that it does not fly. It is only evidence against the conclusion that an injured bird flies. In other words, we do not wish to conclude $\sim flies(X)$ if $injured(X)$, we simply want to prevent a conclusion $flies(X)$.

The *superiority relation* among rules is used to define priorities among rules, that is, where one rule may override the conclusion of another rule.

C. Expressiveness of Defeasible Logic

In our implementation, we use defeasible logic for formulating policies. The main reason is that there is a very efficient method for finding the conclusions of defeasible theories [4]–[6]. The defeasible framework also allows us to express naturally the alternative choices which are common ingredients of policies. But expressiveness of this logic is not clear, while study has shown that the defeasible logic framework can be utilized for a variety of applications.

For the test cases we studied, we encountered a few concepts that do not have a natural translation into defeasible logic; but we were able to formulate them in this framework. As an example we present here how to introduce the "counting" notion in defeasible logic.

D. Counting in Defeasible Logic

Using a concrete example, we will show how to implement counting in the framework of defeasible logic.

Suppose there are thirty-six channels and each party (force) has its own sub-list of available channels. For party A, we use the variables $ChannelA1$, $ChannelA2$, …, $ChannelA36$ such that $ChannelAk$ is true if party A has access to channel k. Similarly, we use the variables $ChannelB1$, $ChannelB2$, …, $ChannelB36$ for party (force) B. The goal is to find out whether there are three channels available to both parties.

We introduce the variable *ChannelSatisfied* which is true if the above condition is satisfied. The defeasible rules we introduce simulate the process of examining the channels 1, 2, …, 36 one by one and the variable *Channelk_n* is true if after examining the channels k has *at least* n channels available to both parties. The rules are as follow:

R1: $ChannelA1$ & $ChannelB1$ → $Channel1_1$

R2: $Channel1_1$ → $Channel2_1$

R3: $ChannelA2$ & $ChannelB2$ → $Channel2_1$

R4: $ChannelA2$ & $ChannelB2$ & $Channel1_1$ → $Channel2_2$

R5: $Channel2_1$ → $Channel3_1$

R6: $Channel2_2$ → $Channel3_2$

R7: $ChannelA3$ & $ChannelB3$ → $Channel3_1$

R8: $ChannelA3$ & $ChannelB3$ & $Channel2_1$ → $Channel3_2$

R9: $ChannelA3$ & $ChannelB3$ & $Channel2_2$ → *ChannelSatisfied*

R10: $Channel3_1$ → $Channel4_1$

R11: $Channel3_2$ → $Channel4_2$

R12: $ChannelA4$ & $ChannelB4$ → $Channel4_1$

R13: $ChannelA4$ & $ChannelB4$ & $Channel3_1$ → $Channel4_2$

R14: $ChannelA4$ & $ChannelB4$ & $Channel3_2$ → *ChannelSatisfied*

⋮

R165: *Channel*34_1 → *Channel*35_1

R166: *Channel*34_2 → *Channel*35_2

R167: *ChannelA*35 & *ChannelB*35 → *Channel*35_1

R168: *ChannelA*35 & *ChannelB*35 & *Channel*34_1 → *Channel*35_2

R169: *ChannelA*35 & *ChannelB*35 & *Channel*34_2 → *ChannelSatisfied*

R170: *ChannelA*36 & *ChannelB*36 & *Channel*35_2 → *ChannelSatisfied*

The template of the rules associated with channel k (except for channels 1, 2, and 36) are as follows:

Rk1: *Channel*(k –1)_1 → *Channel*(k)_1

Rk2: *Channel*(k –1)_2 → *Channel*(k)_2

Rk3: *ChannelA*(k) & *ChannelB*(k) → *Channel*(k)_1

Rk4: *ChannelA*(k) & *ChannelB*(k) & *Channel*(k –1)_1 → *Channel*(k)_2

Rk5: *ChannelA*(k) & *ChannelB*(k) & *Channel*(k –1)_2 → *ChannelSatisfied*

Rules Rk1 and Rk2 guarantee that the number of channels available to both parties in the range {*Channel*1, ..., *Channel*(k –1)} is passed to this stage. Rule Rk3 guarantees that if the *Channel*(k) is available to both parties then it is registered. Rule Rk4 guarantees that if the *Channel*(k) is available to both parties and there is one channel available to both parties in the range {*Channel*1, ..., *Channel*(k –1)} then it is registered in the range {*Channel*1, ..., *Channel*(k)} and there are two channels available to both parties. Rule Rk5 guarantees that if the *Channel*(k) is available to both parties and there are two channels available to both parties in the range {*Channel*1, ..., *Channel*(k –1)} then *ChannelSatisfied* is true.

III. USE CASE ARCHITECTURE

We provide two test cases of implementation of our method. The setting of both cases is the same as defined here:

There are two forces, *Force1* and *Force2*

- *Mission application capacity requirements:* there is a *Pick List* of 36 RF channels and each force has its own sub-list of available channels
  - The goal is to find four channels available to both forces that are also consistent with the other requirements
- *Mission reliability requirements:* each force has access to different packages of adequate IP addresses
  - The requirement is that each force has access to two IP address block
- *Traffic Policy Requirements:* to allow reach back and route back traffic
- Two Paths are available: *Path1, Path2*
- Only one of them can be used

Based on this setting we consider two scenarios. One can be accomplished in one round of negotiation, and the other requires two rounds.

A. A Single Round Negotiation

In this scenario, what two parties (forces) offer is enough to satisfy the requirements; the negotiation engine finds (minimal) subsets for fulfilling all requirements.

The private policy of Force 1:

- Has access to the following ten acceptable channels: Channel5, Channel7, Channel9, Channel12, Channel15, Channel16, Channel17, Channel23, Channel25, Channel32
- Has access to two packages of adequate IP addresses: IPAddressOne1, IPAddressOne2
- If it has access to Video, it cannot use Voice
- If it uses Channel9, it cannot use Video
- If it uses Channel17, it cannot use Voice
- If it uses Channel32, it cannot use Voice
- If it uses Path1, it cannot use Channel9 or Channel15 or Channel32 or IPAddressOne2
- If it uses Path2, it cannot use Channel5 or Channel17 or Channel19 or Video
- If it uses Channel5, it cannot use IPAddressOne2
- If it uses Channel15, it cannot use IPAddressOne1
- If it uses Channel17, it cannot use IPAddressOne2
- If it uses Channe32, it cannot use IPAddressOne1

The private policy of Force 2:

- Has access to the following eleven acceptable channels: Channel4, Channel7, Channel8, Channel9, Channel12, Channel13, Channel16, Channel19, Channel23, Channel25 , Channel34
- Has access to two packages of adequate IP addresses: IPAddressTwo1, IPAddressTwo2, IPAddressTwo3
- If it has access to Video, it cannot use Voice
- If it uses Channel9 it cannot use Video
- If it uses Channel17 it cannot use Video
- If it uses Channel32 it cannot use Voice
- If it uses Path1, it cannot use Channel9 or Channel19 or Channel25 or IPAddressTwo3
- If it uses Path2, it cannot use Channel7 or Channel16 or Channel19 or Video
- If it uses Channel19, it cannot use IPAddressTwo2
- If it uses Channel16, it cannot use IPAddressTwo3
- If it uses Channel25, it cannot use IPAddressTwo2
- If it uses Channe34, it cannot use IPAddressTwo1

The defeasible logic formulation of Force1 policy is as follows, where in rules R8–R17 the set C_1 is defined as

$$C_1 = \{5, 7, 9, 12, 15, 16, 17, 23, 25, 32\}:$$

R1: *ChannelSatisfied* & *IPSatisfied* & *ConnectionSatisfied* & *Path* → *Satisfied*

R2: *IPAddressOne*1 → *IPSatisfied*

R3: *IPAddressOne*2 → *IPSatisfied*

R4: *UseVoice* → *ConnectionSatisfied*

R5: *UseVideo* → *ConnectionSatisfied*

R6: *Path*1 → *Path*

R7: *Path*2 → *Path*

R8–R17: { } ⇒ *ChannelAn*, $n \in C_1$

R18: { } ⇒ *IPAddressOne*1

R19: { } ⇒ *IPAddressOne*2

R20: { } ⇒ *UseVoice*

R21: { } ⇒ *UseVideo*

R22: { } ⇒ *Path*1

R23: { } ⇒ *Path*2

R24: *Path*1 ~> *not–Path*2
R25: *UseVideo* ~> *not–UseVoice*
R26: *Path*1 ~> *not–IPAddressOne*2
R27: *Path*2 ~> *not–UseVideo*
R28: *ChannelA*9 ~> *not–UseVideo*
R29: *ChannelA*17 ~> *not–UseVoice*
R30: *ChannelA*32 ~> *not–UseVoice*
R31: *Path1* ~> *not–ChannelA*9
R32: *Path1* ~> *not–ChannelA*15
R33: *Path1* ~> *not–ChannelA*32
R34: *Path2* ~> *not–ChannelA*9
R35: *Path2* ~> *not–ChannelA*5
R36: *Path2* ~> *not–ChannelA*17
R37: *ChannelA*5 ~> *not–IPAddressOne*2
R38: *ChannelA*15 ~> *not–IPAddressOne*1
R39: *ChannelA*17 ~> *not–IPAddressOne*2
R40: *ChannelA*32 ~> *not–IPAddressOne*1

Priority Relations: R24 > R23; R25 > R20; R26 > R19; R27 > R21; R28 > R21; R29 > R20; R30 > R20; R31 > R10; R32 > R23; R33 > R17; R34 > R10; R35 > R8; R36 > R14; R37 > R19; R38 > R18; R39 > R19; R40 > R18

We should add to the above rules, the rules that guarantee four channels are available to both forces; as they are defined in Subsection II.D.

In the above rules the variable *Satisfied* is true if all requirements satisfied; and the variables *ChannelSatisfied*, *IPSatisfied*, *ConnectionSatisfied*, and *Path* are true if the requirements for channel, mission reliability (IP addresses), connection (voice and video) , and traffic conditions are satisfied, respectively. The defeasible logic translation for Force2 is very similar.

B. A Complexity Calculation

To better understand the subtlety of the above scenario, it would be interesting to see if we want to find the solution through exhaustive search, how many cases we should examine.

Each possible choice can be represented by a quadruple:

(channels, path, connection (video, voice), IPaddresses). The number of choices of channels for Force1 and Force2 are, respectively,

$$\binom{10}{3} = 120 \text{ and } \binom{11}{3} = 165;$$

and the number of possible choices for (path, connection, IPaddresses) for these forces are, respectively, (2, 2, 2) and (2, 2, 3). Therefore, the number of choices for Force1 and Force2 are, respectively,

$$120 \times 2 \times 2 \times 2 = 960, \text{ and } 165 \times 2 \times 2 \times 3 = 1980.$$

If these forces want to compare all their possible choices blindly, then they have to check

$$960 \times 1980 = 1{,}900{,}800$$

cases.

We should mention that our tool solves this problem in a fraction of a second; a testimony to the power of the defeasible logic approach that could avoid such an exhaustive search.

C. A Two Round Negotiation

This scenario is similar to the previous one but, in this case, what two parties (forces) offer initially is *not* enough for satisfying all of the requirements. Therefore the parties need one more round which allows passing this information to each other and new modified offers to achieve the satisfactory conditions. The main difference between this case and the previous one is the initial offers supplied by the parties.

Following the scheme of Fig. 2, we have three policies. The defeasible logic formulation of *Force1 private policy* is as follows (here again in rules R13–R22 the set C_1 is defined as the previous case):

R1: {} ⇒ *StepA*0

R2: { } ⇒ *ChannelA*5

R3: { } ⇒ *ChannelA*7

R4: { } ⇒ *ChannelA*9

R5: { } ⇒ *ChannelA*12

R6: *StepA*0 ⇒ *IPAddressOne*2

R7: *StepA*0 ⇒ *Routbakc*1

R8: *NotSatisfied* ⇒ *not- StepA*0

R9: *ConnectionNotSatisfied* ⇒ *Reachback*1

R10: *IPNotSatisfied* ⇒ *IPAddressOne*1

R11: *IPAddressOne*1 & *IPAddressOne*2 ⇒ *IPSatisfied*1

R12: *Reachback*1 & *Routbakc*1 ⇒ *ConnectionSatisfied*1

R13–R22: *ChannelBn* ⇒ *ChannelAn*, $n \in C_1$

The variable StepA0 is true if it is the first round of Force1. Therefore, the rules R2–R7 provide the initial offer of Force1. The variable NotSatisfied in the rule R8 is true if some requirements are not satisfied and hence it implies rounds after the first round. The rules R9–R12 specify how to modify the next offers when specific requirements are not satisfied. The implication of the rules R13–R22 is that once Force2 offers a channel which is also available to Force1, then Force1 accepts that channel.

Force2 private policy is very similar to Force1. The defeasible logic formulation of *Criteria Policy* is as follows:

R1: *ChannelSatisfied* & *IPSatisfied* & *ConnectionSatisfied* & *Path* → *Satisfied*

R2: *IPSatisfied*1 & *IPSatisfied*2 → *IPSatisfied*

R3: *ConnectionSatisfied*1 & *ConnectionSatisfied*2 → *ConnectionSatisfied*

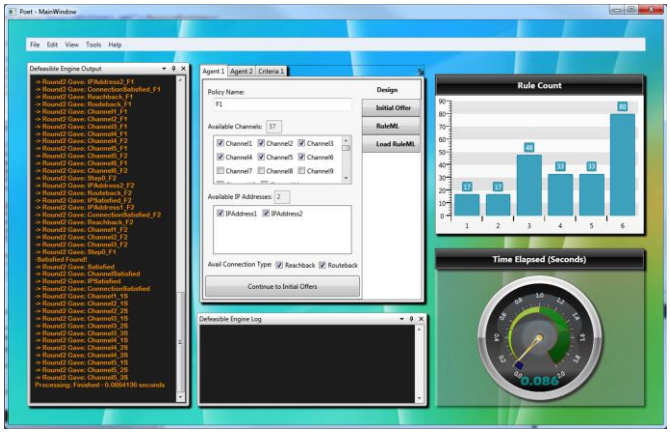

Fig. 3. Policy negotiation tool and editor.

R4: {} ⇒ *ConnectionNotSatisfied*

R5: *ConnectionSatisfied ~> not-ConnectionNotSatisfied*

R6: {} ⇒ *IPNotSatisfied*

R7: *IPSatisfied ~> not-IPNotSatisfied*

R8: {} ⇒ *ChannelNotSatisfied*

R9: *ChannelSatisfied ~> not-ChannelNotSatisfied*

R10: *ChannelNotSatisfied* → *NotSatisfied*

R11: *IPNotSatisfied* → *NotSatisfied*

R12: *ConnectionNotSatisfied* → *NotSatisfied*

Priority Relations: R5 > R4; R7 > R6; R9 > R8

In addition to the above rules, we should add the rules that guarantee each force has access to four channels; as they are defined in Subsection II.D.

E. Developing the Implementation of this Methodology as an Executable Tool

By implementation of the methodology described in the previous section, we have developed a policy negotiation engine. The main elements of this implementation are:

1. A defeasible reasoner engine
   1.1. Coded in C++ and C#
   1.2. The input and output files are in RuleML language
2. An editor GUI for generating and testing the RuleML files of the policies.

The reasoner (DPC) is based on algorithms described in [5], [6]. It accepts the inputs in the form of a RuleML. As it is described in Section II, we are using DPC not only for the combining policies, but also for producing offers in rounds of negotiation as logical conclusions of policies.

TABLE I
NUMBER OF THE RULES AND THE AVERAGE TIME

Number of Rules	Average Time (ms)
6	16.403
11	18.204
22	18.304
48	39.007
106	59.212
153	135.138

The policy editor (Fig. 3) GUI allows the generation of RuleML files simply by entering rules in the form of an easy-to-understand semi-English format. The editor also has access to the DPC reasoner and allows the running and testing of policies once generated or downloaded into the editor GUI.

F. Tool Performance

For benchmarking the DPC tool, we used different sized policies (i.e., different number of rules) and applied the C++ DPC tool and recorded the (average) performance time in milliseconds. This benchmarking is performed on a machine running Windows7 64-bit on an Intel Core i7 CPU clocked at 2.00 GHz with 8GB of installed memory. Table I shows the result.

REFERENCES

[1] L. Gong and X. Qian, "The Complexity and Composability of Secure Interoperation," in *Proceedings of the IEEE Symposium on Security and Privacy*, 1994, pp. 190–200.

[2] 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.

[3] A. Lee, J. Boyer, L. Olson, and C. Gunter, "Defeasible Security Policy Composition for Web Services," in *Proceedings of the fourth ACM workshop on Formal Methods in Security*, 2006, pp. 45–54.

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

[5] F. Vatan and J. Harman, "Efficient Web Services Policy Combination," *NASA Tech Briefs*, pp. 67–68, November 2010.

[6] F. Vatan, and J. Harman, "Efficient Web Services Policy Combination," JPL New Technology Report, NTR 47279, September 2009.

[7] F. Vatan, E. Chow, and G. Paloulian, "Multi-Party Policy Negotiation Engine," JPL New Technology Report, NTR 48042, June 2011.

[8] J. Strassner, *Policy Based Network Management: Solutions for the Next Generation*, Morgan Kaufman, 2003.

[9] R. Yavatkar, D. Pendarakis, and R. Guerin, "A Framework for Policy-based Admission Control," RFC 2753, 2000.