AMPOL: ADAPTIVE MESSAGING POLICY BASED SYSTEM

BY
RAJA N. AFANDI

B.S., Ghulam Ishaq Khan Institute of Engineering Sciences and Technology, 2000

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master's of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2005

Urbana, Illinois

ABSTRACT

Electronic messaging is a critical application that enables electronic communication between a very large collection of principals. However, it is challenged by limits in flexibility, integration, and security. Due to these limits principals are not typically willing to or in some cases not able to accept all types of communications from different parties. This thesis explores the idea of developing adaptive policy based messaging system AMPol, which aims to solve the limitation faced by messaging systems. AMPol approach is to enable parties to advertise communication policies to which a sender can adapt in order to get a message to a recipient on terms acceptable to the recipient. In this work different case studies are described to show how declarative requirements in the form of policy can help system achieve adaptability and flexibility. Especially a case study on email system explores the idea of using cutting edge web technology, web services, as a foundation for expressing and negotiating communications in a large scale system such as Internet email.

In this work we have emphasized on the architectural aspects of the AMPol System and proposed architectural models for Policy Adaptation, Policy Negotiation and System Extension. In future we plan further investigations regarding theory, interoperability, experimentation, and security. Similarly, for wide scale adoption and to illustrate the effectiveness of our proposed approach we need to develop requirements for applications in different specific areas. In this work we have only presented applications in the area of internet email and VoIP communication to support our proposed theory.

DEDICATIONS

This thesis would not have been possible without the support of my parents, relatives and numerous friends. I am also especially thankful to my dear wife who endured this long process with me, always offering moral support and guidance.

ACKNOWLEDGMENTS

This project would not have been possible without the support of many people. First of all, I would like to thank my adviser Carl A. Gunter for his valuable insight and critical feedback which helped me throughout this work. Also thanks to my colleagues and group members, Jianqin Zhang, Munawar Hafiz, Kevin Lux, Kaijun Tan, Jodie Boyer, Zahid Anwar, Moazzam Saleem**,** Himanshu Khurana, and many more for offering me excellent guidance and support.

TABLE OF CONTENTS

LIST OF FIGURES viii

CHAPTER 1: BACKGROUND 1

1.1 INTRODUCTION 1
1.2 REQUIREMENTS 3
1.2.1 Requirements for Policy and Negotiation 4
1.2.2 Requirements for Policy Enforcement and System Extension 5
1.3 MESSAGING MODELS 7
1.3.1 Basic Messaging 7
1.3.2 Four Node Messaging 8

CHAPTER 2: POLICY MODEL 11

2.1 POLICIES 11
2.1.1 Static and Dynamic Policies 11
2.1.2 Policy Model 13

CHAPTER 3: POLICY DISCOVERY AND NEGOTIATION 18

3.1 INTRODUCTION 18
3.2 POLICY MERGING 18
3.3 POLICY QUERY PROTOCOL 19
3.3.1 PQP for Basic Messaging 19
3.3.2 PQP for Four Node Messaging 20

CHAPTER 4: POLICY ENFORCEMENT AND SYSTEM EXTENSION 25

4.1 INTRODUCTION 25
4.2 OVERVIEW OF APPROACH 26
4.3 EXTENSION MODEL 27
4.4 FEATURES AND ISSUES 33

CHAPTER 5: IMPLEMENTATION CASE STUDIES 36

5.1 PUZZLE BASED ANTI SPAM IN WSEMAIL 36
5.1.1 Introduction 36
5.1.2 Architecture and Implementation 39
5.1.3 Discussion 51
5.2 ADAPTIVE VOIP CALL SYSTEM 52
5.2.1 Introduction 52
5.2.2 Background 53
5.2.3 Architecture and Implementation 56
5.2.4 Discussion 60

CHAPTER 6: RELATED AND FUTURE WORK 61

6.1 INTRODUCTION 61
6.2 POLICIES AND NEGOTIATION 61
6.2.1 Trust Negotiation and Policy Management 61
6.2.2 Policy Composition 63
6.2.3 Policy Based Email System 64
6.2.4 Petmail 64
6.3 ADAPTATION AND EXTENSION 67
6.4 FUTURE WORK 69

CHAPTER 7: CONCLUSIONS 72

REFERENCES **73**
APPENDIX A: WSEMAIL-PUZZLE DEMO SCREENSHOTS **76**
AUTHOR'S BIOGRAPHY **82**

LIST OF FIGURES

Fig 1: Basic Messaging Model 7
Fig 2: Four-Node Messaging 8
Fig 3: Examples of APES Rules 14
Fig 4: Policy Merging 19
Fig 5: Policy Exchange and Merging.......... 22
Fig 6: Component Extension Framework 30
Fig 7: Component Invocation 32
Fig 8: RTT Puzzle Examples 39
Fig 9: WSEmail Architecture.......... 39
Fig 10: WSEmail Server Architecture 41
Fig 11: WSEmail-Puzzle Architecture.......... 43
Fig 12: RMTA Policies.......... 45
Fig 13: RMUA Policies 46
Fig 14: SMTA Policies 47
Fig 15: Merged Policies.......... 48
Fig 16: Modified WSEmail Server Architecture 49
Fig 17: Modified WSEmail Client Architecture.......... 51
Fig 18: ACS Architecture 56
Fig 19: ACS Components 59
Fig 20: RMTA Policies.......... 76
Fig 21: RMUA Policies 76
Fig 22: Login Window.......... 77
Fig 23: Message Sending Window 77
Fig 24: RTT Puzzle Download 78
Fig 25: Solving RTT Puzzle 78
Fig 26: Hash Cash Puzzle Download.......... 79
Fig 27: Solving Hash Cash Puzzle.......... 79
Fig 28: Message Received 80
Fig 29: Viewing Message 80
Fig 30: Viewing Error Message.......... 81

CHAPTER 1: BACKGROUND

1.1 Introduction

Messaging in different levels is one of the foundations of distributed system. To satisfy application and security requirements, most entities have communication policies. These policies lead to corresponding protocols. In an open distributed messaging environment, however, communicating entities alter the policies dynamically due to ever-changing requirements and increasing security threats and trust establishment between unacquainted entities. The sending entity needs to identify the policy to set up a secure conversation with the "stranger". This dynamic buildup of trust involves all parties advertising their current communication policies and the interested parties acquiring and adhering to the policies. Adaptation of the communicating parties in response to this policy modification is crucial for the communication to continue.

However, most secure messaging systems are based on fixed protocols; the two sides are limited to those who have known the protocol. Most protocols are not flexible enough to handle changing requirements of the two sides. From the point of the view of the application, most of the systems use messaging policies on incoming and outgoing messages to enforce certain rules for security or service-oriented purposes. But these systems are so brittle that even a slight change in application or environment cripples the whole system.

Adaptation is a relationship between a system and its environment. Systems are often classified as adaptable, which are able to be modified by an external agent or adaptive (also called evolutionary), which are able to modify themselves without external stimuli.

In this work, we propose **AMPol**, *Adaptive **M**essaging **Pol**icy System*, a policy-based adaptive messaging system. The communicating parties check their own environment, determine and exchange the temporal requirements, set up dynamic conversation policies based on static policies by policy advertisement, negotiation and migration. By enforcing the policy, both sides perform agreed operations on the messages, carry out corresponding business process, install and execute designated third-party plug-ins and finally send messages between themselves. Thus, a more flexible and extensible messaging platform is presented to applications.

Adaptation can be viewed in different perspectives:

1. dynamic adaptation of policies enforced by the system
2. adaptation of system interaction protocols
3. adaptation of core application structure

In this work, we analyze these three adaptation perspectives with respect to distributed messaging and service oriented systems. In AMPol these three perspectives can be viewed as Policy Adaptation, Adaptive Policy Discovery & Negotiation, and System Extensibility. This adaptation process in AMPol can generate and enforce dynamic policies to react ever-changing circumstances and requirements in an adaptive manner.

For general purpose, we refer to every node on Internet as an *"entity"* whether it is a client or server or peer, unless specified. We refer to change in functionality either by adding new components or modifying current implementation for adaptive behavior as an *"extensions"*. Generally for change, modification or extensibility of a behavior we use terms "*adaptable", "adaptation" or "adaptive"* interchangeably unless specified.

1.2 Requirements

We have defined following properties for adaptive messaging system:

- **Local and Global Adaptation:** Adaptive system must be able to provide adaptation at multiple levels and in different perspectives. Each node in a system must be able to adapt itself with respect to system extension or requirement specification or policies. Globally the system should be able to coordinate end to end adaptation thus achieving uniformity and conformance with in system entities. In AMPol system each node must be able to adapt to requirements specified in policy and the overall system should maintain its stability and balance.
- **Initiator predictability:** The system must be able to identify the initiator or stimuli responsible for triggering adaptation process. This is important in the AMPol perspective as in messaging system it is required to know the origin of message or the sender; similarly it is important to know who wants adaptation of which entity. This requirement is critical for client driven extensibility and for secure adaptation of a particular entity in the systems. Also system should be able to identify when to change and when not to.
- **Graceful Selection:** If system entities fail to establish a dynamic session then they should be able to gracefully terminate the session and system should maintain its stable state. Similarly the system and its entities should maintain stable state even after the failure of adaptation process.
- **Dynamic Component Update:** To realize the adaptation, the system needs to install the third party's software (plug-ins). An adaptive messaging system should support such feature and make the update as transparent as possible.

- **Secure dynamic protocol:** In a dynamic messaging system, a dynamic protocol will be set up for each session. The dynamic protocol should be reliable, secure, vulnerability – resistant.
- **A coherent, standardized messaging system:** It is usually desirable that all nodes in a system should work in a coherent way and support consistent adaptation throughout the system. Similarly the whole adaptation process, policy specification, merging and negotiation should be uniform. It is preferable to rely on standardized methods and mechanisms that can help achieve interoperability with other standard systems.

We also have following two assumptions:

a) **Communication Protocol**: There is a communication protocol between entities. Based on the protocol, entities can exchange information so set up dynamic policy for a conversation.

b) **Environment Analysis**: Each entity is capable of analyzing the environment, such as network traffic and system performance, to detect potential attacks

1.2.1 Requirements for Policy and Negotiation

For an adaptive messaging system it is very important to have policy language and negotiation mechanisms with certain characteristics, such as:

- The policy language should be simple, well formed and should have well-defined *semantics*, as the meaning of a policy written in specific language should be independent of its implementation.
- The policy language should support basic level logical *operations and standard functions* to represent logical expression of rules within a policy.

- The policy language with standard functions should be *monotonic*. It is usually required for Access Control policy for writing positive rules that grant access to a resources and it does not deny access on providing any additional credentials or identity.
- The policy language should support low level basic data types and it is desirable to have high level *multi-attribute* data types. It should be *extensible* to support new types and functions.
- Policies constructs should be distinct or modular such that policy language should be *composable.* Merged policy should also maintain the semantics and all the above properties.

1.2.2 Requirements for Policy Enforcement and System Extension

Adaptation is a relationship between a system and its environment. Systems are often classified as adaptable which are able to be modified by an external agent or adaptive which are able to modify themselves without external stimuli (also called evolutionary). Our objective is to achieve both Adaptive and Adaptable behavior in a system with following properties, (some of these properties are also identified by [2]):

Independence: Adaptation mechanism should not modify the core base implementation.

Meta-adaptation: Adaptation mechanism should have the ability not only to adapt the system in order to meet requirements and performance goals but to adapt the adaptation process itself in order to ensure that changes are carried out effectively.

Run-time availability: For 7x24 hour availability of the system, adaptation through extensions should ideally be integrated into the system at run time without disrupting the availability of services.

Modular extension: According to the separation of concerns principle, each extension must be implemented as a separate module that can be incrementally added to and removed from the core application.

System-wide consistency: Adapting a system with a single extension goes beyond refining individual components, but also involves a system-wide refinement of multiple components/ clients and interaction with them at the same time. This includes dynamic refinement of interaction behavior with other clients and system components without breaking consistency.

Client-specific customization: The above extensions can exclusively be applied on behalf of the specific clients who have need of these customizations. So a system must provide support for dynamic client-specific extensibility.

Selective combination: Often there is a need for a dynamic and selective combination of extensions on a per collaboration basis. Which subset of extensions must be combined together is dependent on contextual information historically related to the collaboration's message flow in the client and core system.

Scalable: The process of selective combination and system-wide integration of extensions must remain manageable as the number of core component instances and the number of candidate extensions increases.

Performance: The run-time adaptability should not degrade the overall system performance.

Secure: The whole adaptation mechanism should be secure and reliable.

In the above discussion we have identified some basic requirements for adaptive systems, now we need to present a basic messaging model on which we will build on to present a complete model which satisfies above requirements.

1.3 Messaging Models

We focus on two types of messaging model: Basic messaging and Four-Node Messaging.

1.3.1 Basic Messaging

Basic messaging is the base for all messaging systems. (See Fig 1)

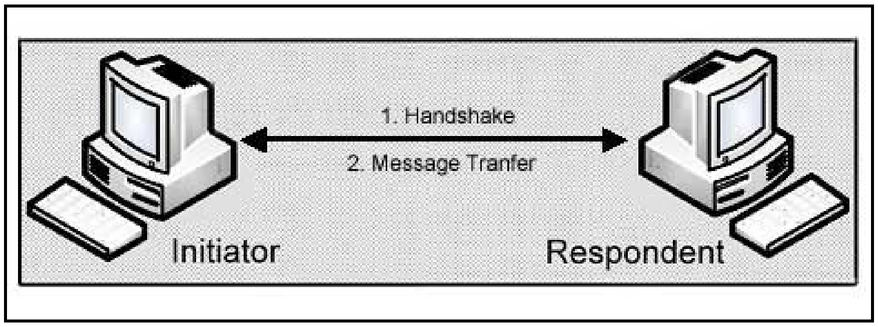


Fig 1: Basic Messaging Model

It has two entities, ***Initiator*** and ***Respondent***. Following assumed protocol, Initiator starts the session to accomplish a particular task, such as downloading a file. Respondent accepts session request from Initiator and cooperate with the Initiator to accomplish the task. Each communication session consists of two phases, Handshake and Message transferring. In the handshake phase, Initiator and Respondent might authenticate each other if authentication is required. They also might exchange the communication requirements, negotiate and come to an agreement of the requirements which both sides should satisfy. In this work, both types of requirements are described in policy and we call the first phase Policy Negotiation. In the Messaging Transferring phase, each side processes the outgoing messages according to the agreed requirements and checks whether the incoming messages satisfy the agreed requirements. In this work, we call this phase Policy Enforcement.

1.3.2 Four Node Messaging

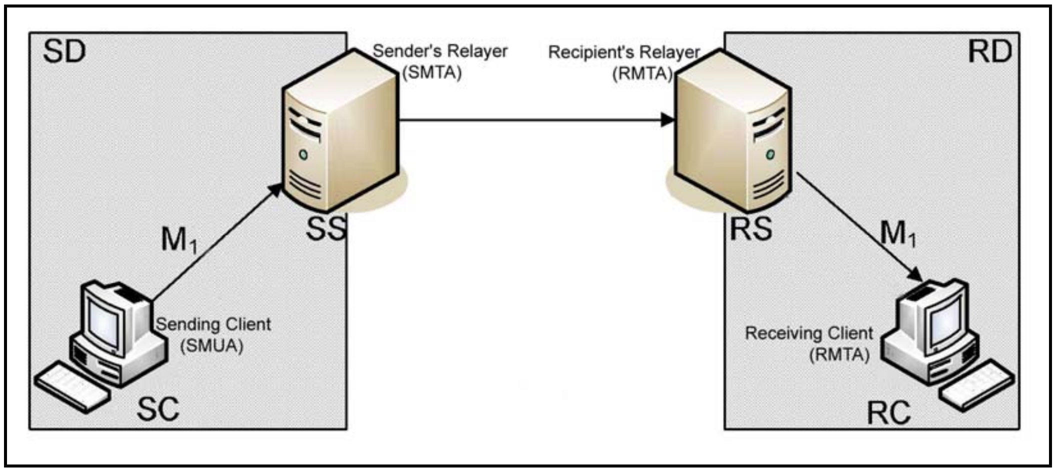


Fig 2: Four-Node Messaging

Four-Node Messaging (See Fig 2) is the typical model of an Email system. It consists of four entities:

1. Sender (Sender MUA or SMUA): The entity that initiates the conversation session composes and submits new messages.

2. Sender's Relayer (Sender MTA or SMTA): The entity that accepts a message and moves it forward towards its destination. The destination may be local or reached via another MTA.

3. Recipient's Relayer (Recipient MTA or RMTA): The entity that accepts a message for local delivery.

4. Recipient (Recipient MUA or RMUA): The entity that receives the message.

In the following, we shall use terms defined in RFC2476 [1] to denote the nodes. In a general Email messaging system, there are more than two MTAs. For simplicity, we only focus on the scenario of two MTAs.

The task of the Four-Node messaging model is to transfer a message from SMUA to RMUA. Because they may not be online at the same time and can not talk to each other directly, the messages must be relayed by intermediary. SMUA sends the message to SMTA; SMTA stores and forwards to RMTA; RMUA delivers the message from RMTA. Each conversation above is an application of Basic model. Additionally, because SMUA's message reaches RMTA and RMUA eventually, there are two virtual conversations, between SMUA and RMTA and between SMUA and RMUA. In general email systems, these two virtual conversations are ignored. RMTA and RMUA can do nothing on SMUA but accept the message. This vulnerability is one of the main reasons for SPAM. While filter based approaches do provide mechanism for restricting certain messages to be placed in user's message in-bin, but this approach does not provide any mechanism to establish a virtual communication channel between SMUA and RMUA/RMTA. Due to the absence of this virtual agreement between SMUA and RMUA/RMTA, SMUA is not able to conform its message to the requirements of RMUA/RMTA and it usually result in a failed message delivery. Thus filter based solutions do not prevent spam as much as they interfere with every-day email communications.

In this work, we will address how to implement the Handshake (Policy Negotiation) and Message Transferring for the two virtual conversations we have discussed above, and in the end we will emphasize on system extension for adaptive policy enforcement.

CHAPTER 2: POLICY MODEL

2.1 Policies

Security *policy* is a statement of what, and what is not, allowed. In this work, we treat it as a set of rules or constraints which describes entity's security requirement for communication. For example, Alice may accept the message from Bob only if it is signed by Bob using his certificate; a SMTA will not relay a message from SMUA unless the SMUA offers the solution of a Puzzle [32], [33], [34], [35] problem assigned by SMTA.

In AMPol, policy is the key point to describe the adaptive actions. Each entity of the conversation takes adaptive actions based on the policies. How to guarantee the policy to describe the system's requirement and adaptive actions accurately and how to generate the effective policies? are significant issues. We will be addressing these issues in sections below.

2.1.1 Static and Dynamic Policies

Entities have policies for creating a conversation sessions with other clients. Entities also have rules to instruct other clients to set up dynamic policies for each conversation session. For a particular conversation session, as the environment and the characteristic of client varies, the security policies are also different. For example, in a trusted domain, entities may not authenticate each other and messages can be sent in plain or with weak encryption. While in an open domain, entities must authenticate each other; confidential messages must be encrypted strongly; some additional methods to counter particular attacks, such as DoS, are introduced. Additionally, for different entities, one entity can have different policies.

So, we divide policies into groups of *static policy* and *dynamic policy.* Static policy is a policy framework for any conversation session. It defines some general rules and constraints for communication. Static policy also defines rules to create and negotiate dynamic policy. For example, entity Alice defines the following static policy:

1. Every incoming message must be signed by sender using sender's certificate;
2. Every outgoing message must be encrypted using designated algorithm and pre-shared key;
3. If an incoming message is to be relayed and the sender is unknown, some mechanisms, say Puzzle [34] or Cookie [36], must be applied to reduce the risk of being compromised by a DoS attack.

Before the message is transferred, the two entities involved in the communication must set up dynamic policy which should satisfy the requirements of each participant in a specific communication sesion.

Dynamic policy is a set of rules and constraints on a particular (specific) conversation session. It is based on static policy, but it is specific for every conversation session. Entities generate dynamic policy based on static policy, environment and policy negotiation. Environment refers to the entity system's status and the network. By analyzing some parameters of the network or system, an entity can detect some potential attack. The characteristic of the other conversation entity is another element of environment.

Communication entities exchange static policies to get each other's requirements for the conversation. They make an agreement on the dynamic policy. We call such procedure a *policy negotiation*.

During the process of policy negotiation, both sides will get a couple of policies. They need to be merged to a single policy set. Some of the collected policies may be conflicting or inconsistent with each other. We need some mechanisms to reconcile the inconsistency, and to detect and solve conflicts. We call this mechanism *policy merging*.

2.1.2 Policy Model

- Action Rules (**AR**): Action rules describe the operations required for the Confidentiality, Integrity and Availability of communication. For each operation, we assume the subject is sender/receiver and the object is the message. Each action rule is a pair of action name and action property: (***A***, ***P***).
  - ***A*** is a set of possible operations on message. For example, {*Attachment, P*uzzle*, E*ncryption, *S*ignature}. Here we use the first capital letter of the operation name to denote a particular operation.
  - ***P*** is a descriptor of the details of the operation. For example, for Encryption, it indicates the algorithm, key size and other parameters used in encryption; for Puzzle, it indicates what type of Puzzle will be applied, RTT or Hashcash. ***P*** also contains the address where sender/receiver can get the third-party plug-in. This is an important feature for adaptive system because sender/receive needs to install plug-in to satisfy unknown side's requirements. To be compatible with Web Service specification, ***P*** is formatted as a URI in the implementation.

  AR: $\{(a, p) \mid a \in \mathbf{A}, p \in \mathbf{P}\}$

In web service based messaging system, there are four common actions. They are ***A***ttachment, ***P***uzzle, ***E***ncryption and ***S***ignature. We call these rules **APES**. *A*ttachment is the actions on attachments. Because attachments are a possible source of virus, rules for attachments are necessary. For Attachment, the corresponding property descriptor ***P*** would be filter patterns for block/filter the particular. Figure 3 below contains some examples of Action Rules:

(Attachment, ua), ua=
"http://seclab.cs.uiuc.edu/AMPol/AR=A&pattern=RemoveEndwithPif"
The attachment in the message should be removed if its name ends with "pif".

(Puzzle, up), up=
"http://seclab.cs.uiuc.edu/AMPol/AR=P&type=RTT&download=
http://seclab.cs.uiuc.edu/Plugin#RTT"

The other communication initiator is required to resolve RTT puzzle; initiator can get RTT plug-in from "http://seclab.cs.uiuc.edu/Plugin#RTT"

(*E*ncryption, u_e), u_e=
"http://seclab.cs.uiuc.edu/AMPol/AR=E&Alg=AES&keysize=1024"
The message should be encrypted by AES, and the key size is 1024.

(Signature, us), us=
"http://seclab.cs.uiuc.edu/AMPol/AR=S&Alg=DSA&keysize=512"
The message should be signed by DSA, the parameter is…

Fig 3: Examples of APES Rules

A is an extensible set. We can add new type of actions to the set to satisfy new security requirements. Because ***P*** is a URI style descriptor, it is flexible to construct various properties for any action.

- Rules Set (**RS**): A policy (Static or Dynamic) is a set of rules. Each rule is a pair of Application (**AP**) and Action-Rules (**AR**), denoted as (**AP**, **AR**). RS is the set of all rules.

RS: $\{(ap, ar) \mid ap \in AP, ar \in AR\}$

Application (**AP**) indicates the usage of the policy. In messaging system, there are two typical types of usage for entity. **AP** = {***I***ngress, ***E***gress}. *I*ngress policy is applied on incoming messages; ***E***gress policy is applied on out-going messages.

- Static Policy (**SP**): **SP** defines a set of rules to be enforced during the communication process. **SP** is a subset of RS.

  SP$\subseteq$**RS**

  SP defines each entity's basic policies for incoming and outgoing messages. It is the base for generating dynamic policy. Following are some examples of Static Policy (The property descriptor of action is same as the above example):

  SP$_1$ = {(*I*ngress, (*S*ignature, u_s)), (*I*ngress, (*P*uzzle, u_p)), (*I*ngress, (*A*ttachment, u_a)), (*I*ngress, (*E*ncryption, u_e))}.

  SP$_1$ defines a set of rules for ingress messages. Sender of the message is required to sign, encrypt the message. Sender may or may not give a solution of Puzzle. If sender attaches a file whose file name extension is "pif", the attachment will be filtered out by the recipient. It is also mandatory.

- Dynamic Policy (**DP**): The data model of a dynamic policy is same as static policies. **DP** is a subset of **RS**. **DP** is generated during the process of policy negotiation.

  DP$\subseteq$**RS**

- Operation on Policy: For generating Dynamic Policy, we define a group of operations on policies.

- **Create** an empty policy:

  Create: $\rightarrow$ **RS**

- Add a rule to a policy (Rule Set):

  Add: **AP** × **AR** × **RS** → **RS**

  Semantic: $ap \in \mathbf{AP}$, $ar \in \mathbf{AR}$, $rs \subseteq \mathbf{RS}$, **Add** $(ap, ar, rs) = rs \cup \{(ap, ar)\}$

- Remove a rule from a policy (Rule Set):

  Remove: **AP** × **AR** × **RS** → **RS**

  Semantic: $ap \in \mathbf{AP}$, $ar \in \mathbf{AR}$, $rs \subseteq \mathbf{RS}$, **Remove** $(ap, ar, rs) = rs - \{(ap, ar)\}$

o Summary:

- **A**: Action set, a set of possible operations on message.

  {*E*ncryption, *S*ignature, *P*uzzle, *C*…, *A*ttachment…};

- **P**: Property of action, a set of URI style descriptor.

- **AR**: Action Rule, a pair of (**A**, P)

- **AP**: Application set, the set of usage. Currently, we only concern policies applied on incoming and outgoing message.

  {*I*ngress, *E*gress}

- **N**: Necessity set. Used with **AR**, indicating whether the rule is *m*andatory or *o*ptional.

  {*m*andatory, *o*ptional}

- **RS**: a set of 3-tupel (**AP**, **AR**, **N**). RS is the set for all rules.

  $\{(ap, ar) \mid ap \in AP, ar \in AR\}$

- **SP:** Subset of **RS**.

For example: {(*I*ngress, (*S*ignature, u_s)), (*I*ngress, (*P*uzzle, u_p)), (*I*ngress, (*A*ttachment, u_a),), (*I*ngress, (*E*ncryption, u_e))}.

- **DP**: Subset of **RS**.

In this work, for simplicity, we have based policy model on APES action rules. Surely new action rules can be added (e.g. rules for QoS, protocol versions, Bayesian anti-spam filters etc), but currently we are focusing on application areas e.g. security and internet email, for which we think APES rules are sufficient to represent application requirements in the form of AMPol policy model. The AMPol policy model is extensible enough to support other action rules and in future we plan to propose a generic policy model for AMPol which can support action rules to model requirements of any application domain.

CHAPTER 3: POLICY DISCOVERY AND NEGOTIATION

3.1 Introduction

Entities need to make an agreed dynamic policy before sending the message. We call this process Policy Negotiation. We define the following principals for policy negotiation:

1. **Priority Principal.** If the static policies from initiator and respondent conflict each other, the respondent's has higher priority. If initiator can not yield, the negotiation fails and the conversation stops.
2. **Effort Symmetry.** In order to prevent DoS attack or other similar attacks, initiator should raise the dynamic policy at first. In this way, if the initiator wants the respondent to consume some resource, it needs to consume its own resource at first.

In Policy Negotiation, we need protocols to exchange static policy and dynamic policy. We also need a policy merge method to generate dynamic policy based on static policies.

3.2 Policy Merging

Given two SP's from initiator and respondent (See chapter 2), we merge them into a dynamic policy. If one rule from respondent conflict with the other rule from initiator then the latter is removed. Following is the pseudo code of the algorithm of policy merge.

Merge: RS × RS → RS. Given sp_i, $sp_r \subseteq$ **RS (**sp_i denotes the static policy from initiator; sp_r denotes the static policy from respondent.

```
Merge ($sp_i$, $sp_r$)
    $dp$ = Create();
    for each ($ap$, $ar$) in $sp_r$
        $dp$ = Add ($ap$, $ar$, $dp$);
    for each ($ap$, $ar$) in $sp_i$
        $dp$ = Add ($ap$, $ar$, $dp$);
    for each ($ap_i$, $ar_i$) in $sp_i$
        for each ($ap_r$, $ar_r$) in $sp_r$
            if ($ap_i$, $ar_i$) conflict with ($ap_r$, $ar_r$) then
                $dp$ = Remove ($ap_i$, $ar_i$, $dp$)
    return $dp$
```

Fig 4: Policy Merging

3.3 Policy Query Protocol

Policy Query Protocol (PQP) is used for Policy-based messaging system. PQP specifies the stages of communication of APES rules. We discuss the protocols for Basic Messaging and Four-Node Messaging respectively.

3.3.1 PQP for Basic Messaging

The parties involved in the basic messaging system can be classified as the initiator and the respondent of the protocol. We denote the ingress and egress policies of initiating party as Π_{Ini}^{I} and Π_{Ini}^{E} respectively. Similarly the ingress and egress policies of responding party are denoted as Π_{Res}^{I} and Π_{Res}^{E} respectively. Here are the message exchanges.

Step 1: Initiator (denoted as Ini) wants to communicate with respondent (denoted as Res). It queries the ingress policy of respondent.

Ini $\rightarrow$ Res: Query for Π_{Res}^{I}

Step 2: Respondent returns the static policy, together with the validity period and a signature (signature is optional).

Res$\rightarrow$ Ini: Π_{Res}^{I}

Step 3: Initiator verifies the signature and then merges its own egress static policy and respondent's ingress static policy. This generates the dynamic egress policy for initiator $\Pi_{Ini}^{E'}$.

$\Pi_{Ini}^{E'} = \Pi_{Ini}^{E} \oplus \Pi_{Res}^{I}$ [$\oplus$ denotes the merging operator]

The message for respondent is prepared by conforming to dynamic policy. It also pushes to the respondent this new dynamic policy.

Ini $\rightarrow$ Res: $\Pi_{Ini}^{E'}$ || Message complying with APES rules described by $\Pi_{Ini}^{E'}$

Step 4: Respondent evaluates the dynamic policy $\Pi_{Ini}^{E'}$, to see whether it conforms to its static policy. If not, respondent discards the message immediately. Otherwise, it verifies whether the message conforms to the dynamic policy. If the message conforms to the dynamic policy, it is accepted and an Acknowledgement (ACK) is sent to initiator.

Res$\rightarrow$ Ini: ACK

3.3.2 PQP for Four Node Messaging

We illustrate the protocol in relation to the mail-messaging scenario where the SMUA and RMUA are associated with SMTA and RMTA. A message M is to flow from SMUA to RMUA via SMTA and RMTA. The four entities involved here have ingress and egress policies specified statically in the APES format. We denote the ingress policy specification of receiving client (RC) as Π_{RC}^{I}, the egress policy of sending server as Π_{SS}^{E} and so on. It may be argued that some of the combinations are meaningless, for example the egress policy for a sending client, Π_{SC}^{E}, because this implies that a client is imposing a policy on itself. However, such a policy can be existent as the policy is dynamically negotiated and the sending client wants to comply with the ingress policy of the receiving client and server, and enforces them as the egress policy upon itself.

In the initial stage, some optional policy exchange occurs between the sending client and the receiving client and their corresponding servers. These are as follows:

- **Pull egress policy.** The sending client pulls the egress policies from the sending server and merges them with its existing APES policies. A policy merge is defined as a union of the values of the APES policies. The policies are considered to be non-overlapping meaning the sending server and the sending client does not define non-conforming value for P etc. The result of the merge defines an egress policy for the sending client and we denote it here as,

  $$\Pi_{SC}^{E\prime} = \Pi_{SC}^{E} \oplus \Pi_{SS}^{E} \qquad [\oplus \text{ denotes the merging operator}]$$

- **Push ingress policy.** The receiving client pushes its ingress policies to the receiving server and the receiving server merges them with its existing APES policies. This gives the merged ingress policy for the sending server which is denoted as,

  $$\Pi_{RS}^{I\prime} = \Pi_{RS}^{I} \oplus \Pi_{RC}^{I}$$

  The equilibrium state reached after exchange of these policies is shown in figure 5. The reason for this exchange is to have better performance as it reduces the number of queries for advertisement that a client has to make. Instead of making two queries, the sending client now makes only one query to the receiving server and gets the merged policies for the receiving client. Ignoring this exchange has no impact on the overall protocol except for the performance.

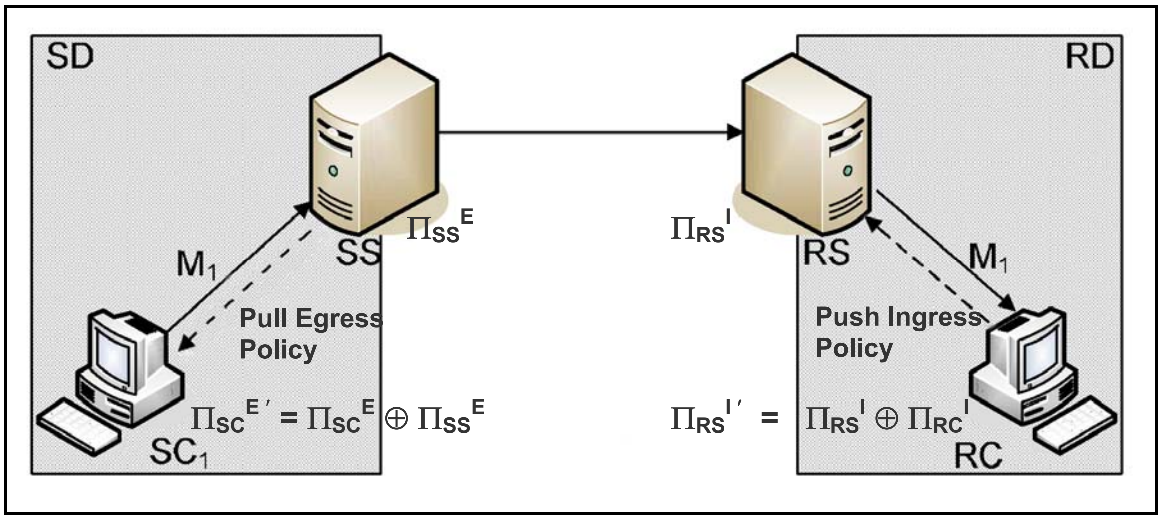

SD
RD
Π_{SS}^{E}
Π_{RS}^{I}
SS
RS
M_1
M_1
Pull Egress Policy
Push Ingress Policy
$\Pi_{SC}^{E'} = \Pi_{SC}^{E} \oplus \Pi_{SS}^{E}$
$\Pi_{RS}^{I'} = \Pi_{RS}^{I} \oplus \Pi_{RC}^{I}$
SC_1
RC

Fig 5: Policy Exchange and Merging

The policy query and execution phase can now be described.

Step 1. SC wants to communicate with RC. SC queries the advertised ingress policy for RC and RS and sends this query to SS.

SC $\rightarrow$ SS: Query for Π_{RS}^{I} and Π_{RC}^{I}

Step 2. SS relays the request to RS.

SS $\rightarrow$ RS: Query for Π_{RS}^{I} and Π_{RC}^{I}

Step 3. RS sends the merged policy $\Pi_{RS}^{I'}$ to SS.

RS $\rightarrow$ SS: $\Pi_{RS}^{I'}$

Step 4. SS sends the merged policy $\Pi_{RS}^{I'}$ to SC.

SS $\rightarrow$ SC: $\Pi_{RS}^{I'}$

Step 5. SC sends messages complying with $\Pi_{RS}^{I'}$ via SS. This goes through RS to RC and RC accepts it because it is compliant to its ingress policy.

SC $\rightarrow$ RC: Message complying with APES rules described by $\Pi_{RS}^{I'}$

Step 6. RC sends acknowledgement to SC via RS and SS. SC receives ACK and is assured the policy is correct.

RC $\rightarrow$ SC: ACK

If the policy merging stages described before are non-existent, then step 2 and 3 become as follows,

Step 2a. SS relays the request to RS.

SS $\rightarrow$ RS: Query for Π_{RS}^{I} and Π_{RC}^{I}

Step 2b. RS relays the request to RC.

RS $\rightarrow$ RC: Query for Π_{RS}^{I} and Π_{RC}^{I}

Step 3a. RC sends its ingress policy Π_{RC}^{I} to RS.

RC $\rightarrow$ RS: Π_{RC}^{I}

Step 3b. RS merges the ingress policies and gets $\Pi_{RS}^{I\prime}$ .It sends the merged policy $\Pi_{RS}^{I\prime}$ to SS.

RS $\rightarrow$ SS: $\Pi_{RS}^{I\prime}$

The policies have temporal value associated with them, which denotes the lifetime. After that time is expired, policies are renegotiated. However, say RC changed Π_{RC}^{I} and pushed that to RS to have a new value of $\Pi_{RS}^{I\prime}$**.** SC still has previous binding, as it does not get any invalidation message. When it tries to communicate, the message is rejected and SC queries again to get the new binding.

A more conservative approach is to have each dynamic policy binding valid for the session only. A new session would always have to acquire the APES policies. This makes policy alteration much easier as the policies are always current. However, it has performance overhead associated with it.

CHAPTER 4: POLICY ENFORCEMENT AND SYSTEM EXTENSION

4.1 Introduction

We need a mechanism by which the affected nodes of the system can dynamically reconfigure or adapt themselves so that they can work properly and also the clients of the system can be informed so that they can abide by these new requirements in advance. In some cases system itself or clients have to add new components in order to support these changing requirements. So in order to support adaptive message communication in AMPol, we also need a secure and reliable model for adapting or extending the behavior of system itself. Adaptation or extension in system functionality is triggered by AMPol messaging policies which may require new plug-ins/components to fulfill the requirements for successful conversation setup or message delivery. We have proposed a policy driven framework for managing and extending these plug-ins/components for functional adaptability of a particular system node in AMPol. This extension framework provides a functionality to achieve extended behavior which can be used for successful enforcement of AMPol messaging policies. Although this framework is generic enough for general purpose component management and extensions, but we will focus toward AMPol messaging specific requirements.

Adaptation is a relationship between a system and its environment. Systems are often classified as adaptable which are able to be modified by an external agent or adaptive which are able to modify themselves without external stimuli (also called evolutionary).

Our objective is to achieve both Adaptive and Adaptable behavior in a system with properties discussed in requirement section.

4.2 Overview of Approach

The basis of AMPol system is dynamic component composition. For achieving functionality adaptation, applications are made up of small components which are brought in at run-time from the network when needed, executed and discarded after use (if they are not required for further use). Which components are used to achieve a particular functionality depends on the messaging policies, run-time execution environment, extension policies and system constraints, thus under different parameters different component will be used, hence achieving adaptive and extensible behavior.

Sometimes support for dynamic and client-specific customization is also required. The AMPol extension model provide this by a minimal functional core implemented as a component-based system and an unbound set of potential extensions that can be selectively integrated within this core functionality. An extension to this core may be a new component, due to new requirements of end client (in messaging policy).

A component is an independently deployable unit of composition with well-defined interfaces and explicit dependencies. A component may encapsulate state and provide interfaces which define services the component offers to clients, which also forms a contract between a client and a component. Component technology is well suited in realizing reconfigurable and extensible systems. Component frameworks proved independence which means that components are inherently loosely coupled and any dependencies are explicit and visible which eases the process of adaptation at run-time. In general, a component configuration can be manipulated dynamically by adding, removing and replacing components and changing connectivity.

The AMPol component extension model (framework) defines a software architectural model in which the components represent the most variable elements of the domain. A component-based application can thus be instantiated from the component framework by simply providing the specific components needed for the particular application. Customization and extension of the application is then achieved by replacing components with more suitable ones. The novelty of this component model is that it allows policy driven selective combination of extensions on a per collaboration basis, enabling easy client-specific customization and extensions. It also supports self adaptable adaptation mechanism, which is called meta-adaptation. In the related work section we have done a detail comparison of our proposed model with other related efforts.

This section presents the component selection and extension model, its implications, and how it can be realized to build adaptable messaging system. We also present a notion for the customization process of a component-based application as a selective combination of one or more extensions into a minimal core system.

4.3 Extension Model

In AMPol extension model, the new functionality module which is needed to be added is referred as *"extensions"*, while the module which has already been integrated is referred as *"component"* and accordingly the Extension Manager and Component Manager manages these entities.

In AMPol, extensions are like normal components with some of the following attributes:

- **Meta Specifications:** This information is used in the evaluation of extension policies and for verification of extension by Extension Validator.
    - Type: It describes the nature of extension as either a Java plug-in applet or a windows dll or a web service etc.
    - Dependencies: This attribute specifies the dependency on other components.
    - Constraints: It describes the dependencies with respect to operating environment, hardware and other technical constraints etc.
    - Functional specifications: High level description of services provided by the component with input and output formats. Usually it represents interface definition (e.g. idl or wsdl file).
    - Owner Information: It contains a certificate of the owner of a component.
    - Identifier: It is used for identification along with version number.
- **State:** It represents a persistent data of a component for maintaining state.
- **Executable:** Binary executables.

Figure 6 gives a high level overview of component and extension management framework. The AMPol messaging policy (resultant merged dynamic policy from all nodes) is fed to the ***Component Manger*** through ***Component Evaluator***, which evaluates the policy and outputs a list of required components needed for successful evaluation of a policy. The ***Component Selector***, on the basis of its ***Component Selection Policies***, searches for required components from the ***Component Repository***. Currently the framework only provides local searching from a repository which is locally managed. We are looking at some global component searching mechanisms, by which we can search for a component on Internet through different other techniques also. The Component

Repository maintains components with their meta-specification, state, code and meta-searching information. The meta-searching information is used for efficient searching of a component. If a Component Selector is able to find a component, then it asks a ***Component Loader*** to load a corresponding component. Component Loader decides among different ***Execution Strategies*** to load a component and decision is based on a type of component, current operating environment, available resources etc.

The Component Loader chooses from following Execution Strategies:

- **Standalone/Remote:** The Component is loaded either remotely on any other machine (Remote) or as a Standalone component: locally in a different address space as a completely independent process (as incase of remote object executed locally).
- **Sand Box [37]:** In this approach the component is executed in a restricted and resource safe environment with minimal possible permissions. The sandbox environment ensures component code is unable to reference memory or resources that are not made available to the environment.
- **Internal:** The Component is loaded in a same address space of current process or in a new thread of a local parent process. This is not a recommended strategy due to higher security risks. It is only recommended when component is from trusted third party.

New strategies and corresponding execution logic can be easily added by extending the behavior of the Component Loader.

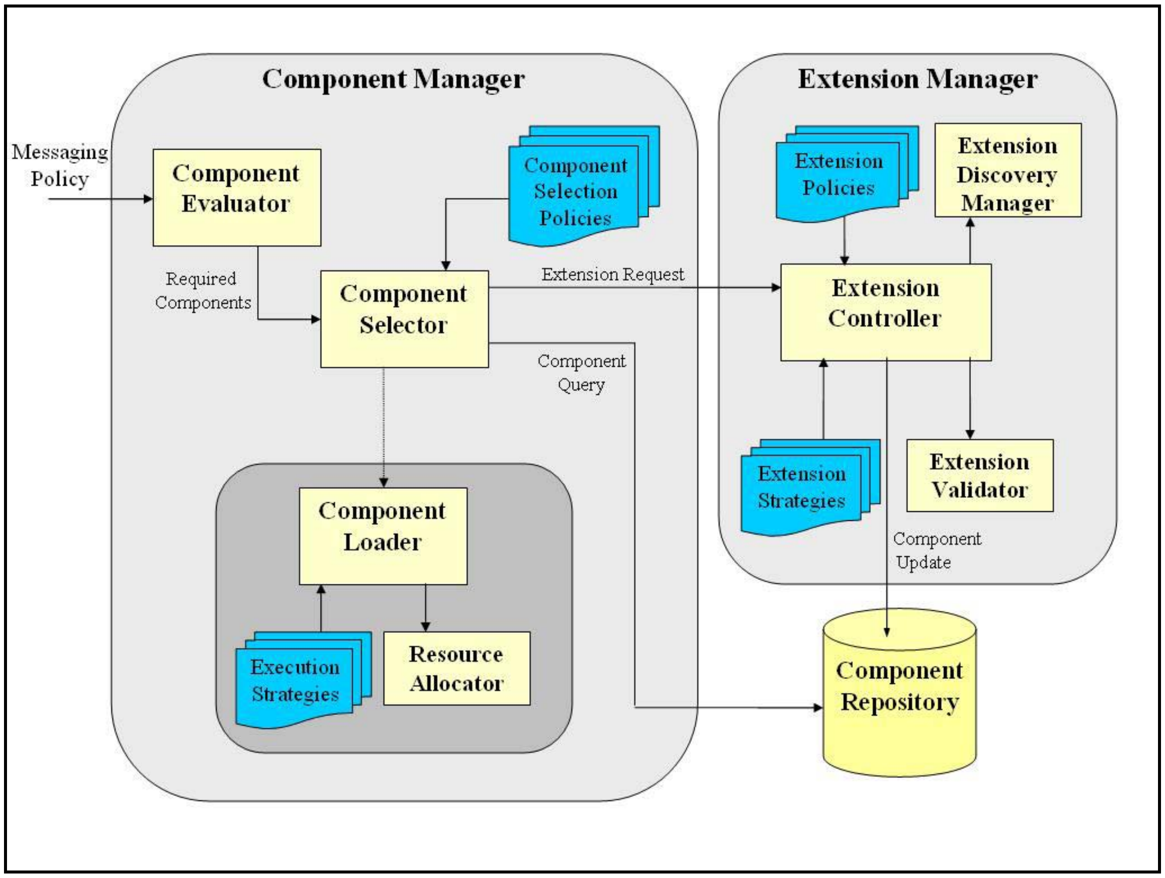


Fig 6: Component Extension Framework

Using a specific type of execution strategies, a Component Loader accordingly loads and executes the component. There are different types of extensions/components and accordingly different types of execution mechanisms which depend upon the component type and specification:

- **Volatile/Stateless:** This type of execution component does not maintain or keep state. Component execution without state especially for single call functionality is also called volatile execution.
- **Persistent/State full:** This type of execution component does maintain or keep state among multiple calls. This execution mechanism is used for client-specific customization.

- **Transient/Disposable:** In this execution, the components are disposed or deleted from repository or system after execution. These components are for one-time usage and keeping them in the system may be vulnerable to the system as hackers can use them for any exploit etc.
- **Remote:** Components are executed remotely as in Remote Execution Strategy.

Component Loader is also responsible for managing these types of executions. ***Resource Allocator*** is responsible for allocating resources for component execution, for example allocating a remote machine for executing a component remotely etc.

If a Component Selector is not able to find a required component, then it asks the ***Extension Manager*** to download that component. The Extension Manager is responsible for extending current functionality/capability of the system by extending the component base. The ***Extension Controller*** controls the whole extension process which is driven by ***Extension Policies***. These policies define criterions for the download and execution of extensions, thus limiting certain type of components to be added as a possible extension. Extension Controller support different types of ***Extension Strategies***, which specifies the extension download and verification mechanism. Some possible strategies are:

- Extension from Trusted Third Party
- Verifiable Extensions
- Un-Trusted Extensions
- Local Extensions

The ***Extension Validator*** verifies the extension incase if verification is required. After the extension is downloaded and verified, it is then registered and stored in the Component Repository so that a Component Selector can search and load it.

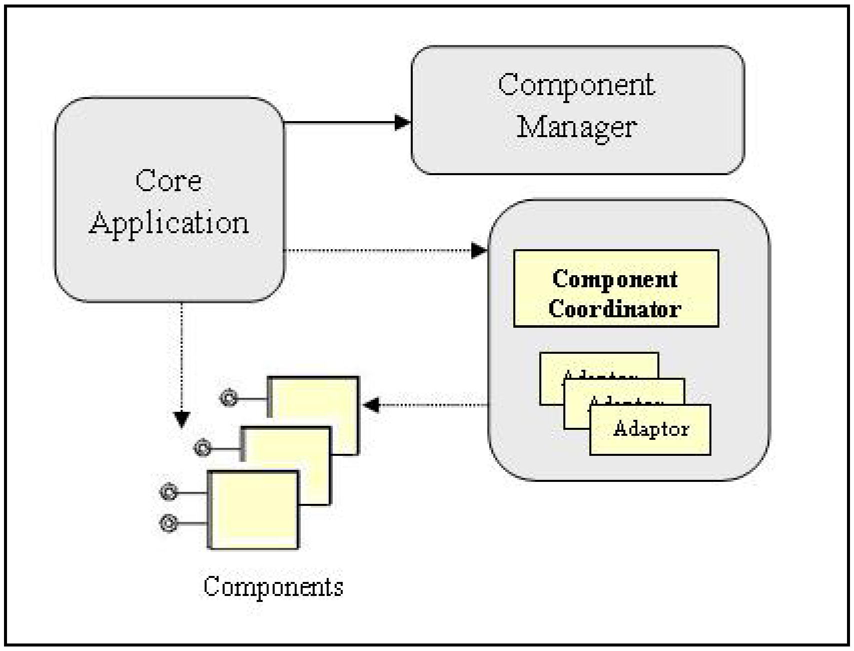


Fig 7: Component Invocation

A component is ready to service a request after it is loaded by a Component Loader. So a core application (the AMPol core component) whenever require a services of a extended component, it asks the ***Component Coordinator.*** The Component Coordinator invokes component interface and accordingly replies with results. The core application can also directly invoke the component interface if it supports the invocation logic. Sometimes if a component interface specification does not exactly conform to the core application or the messaging policy requirements, then the Component Coordinator uses ***Adaptors*** to transform data formats between application and components. These Adaptors are also type of components which can be managed and extended just like other extensions and components.

4.4 Features and Issues

In this section we will discuss that how AMPol extension model fulfills the requirements identified in the background chapter.

We have argued that an adaptive messaging system should have the ability not only to adapt the system in order to meet the requirements and performance goals but also should have the ability to adapt the adaptation process itself in order to ensure that changes are carried out effectively. This framework enables our goal of ***meta-adaptation***: adaptation in the adaptation process itself. AMPol Extension framework itself is composed of plug and play components which can be replaced to achieve adapted or extended functionality. Similarly, functionality of different core components of the framework can be inherited and new customized extensions can be developed. In addition, Component Selection Policies, Extension Policies, Execution and Extension Strategies can be dynamically added or updated to achieve modified component and extension management behavior. We believe this added dimension of flexibility is essential in systems which are to adapt themselves in the face of changing operating conditions.

In accordance with the separation of concerns principle, each extension in AMPol extension framework is implemented as a separate component that can be incrementally added to and removed from the core application hence ***modular, incremental and manageable functionality extension*** can be achieved.

The AMPol extension framework presents an ***independent extension mechanism*** on top of the core implementation and thus does not modify the core implementation. Due to this independence, extensions are integrated into the system at run time without disrupting the availability of other services, which ensures the ***24x7 runtime availability*** of the system, even in the process of adaptation. Similarly due to above discussed independent and modular extension mechanism, the process of selective combination and incremental integration of extensions remain ***manageable and scalable*** as the number of core component instances and the number of candidate extensions increases.

As defined earlier, in the AMPol extension model, extensions are loosely coupled, well defined components having explicit dependencies and due to overall modular structure of extension framework; it is easy to maintain ***system wide consistency***. As we discussed earlier that adapting a system with a single extension goes beyond refining individual components, but also involves a system-wide refinement of multiple components and interaction with them at the same time in order to avoid inconsistency. In the current scope we have limited our work to extensions for which there is no or minimal effect on other components and their interaction. However, with these relatively simple components and extensions we can easily maintain system wide consistency. In the future we will be looking into scenarios for maintaining consistency incase of complex system-wide interactions and dependencies of extensions.

The AMPol extension model provides ***client-specific extensibility***. Extensions can exclusively be applied on behalf of the specific clients who have the need of specific customizations by specifying them as a requirement in a messaging policy. Currently the model only supports stateless client-specific extensions and their customizations. Further work is required to manage complex state-full per session or multi session client-specific dynamic extensibility and customization.

The Component Selector provides a policy driven support for ***dynamic and selective combination of components*** on a per collaboration basis. Which subset of extensions must be combined together is dependent on the messaging policies, run-time execution environment, extension policies and system constraints.

In this work we have not addressed security vulnerabilities and threats to our proposed extension model. In future we are planning to provide a threat model along with possible security solutions to tackle different types of attacks. Similarly in future we plan to critically evaluate the overall performance and reliability of our framework and will propose some enhancements for better efficiency and performance. The performance tests of our proposed extension manager (simplified and limited implementation of plug-in architecture), which is implemented in WSEmail1.0, shows promising results [16].

CHAPTER 5: IMPLEMENTATION CASE STUDIES

5.1 Puzzle Based Anti Spam in WSEmail

5.1.1 Introduction

WSEMail [16] is an electronic mail system which is developed as a family of web services. As we know, the advent of web services has created the foundation for highly interoperable distributed systems to communicate over the Internet using standardized protocols and security mechanisms. WSEmail is basically an attempt to fully exploit these mechanisms and, to analyze and redesign older systems and protocols to see whether they can benefit from the new architecture and technologies.

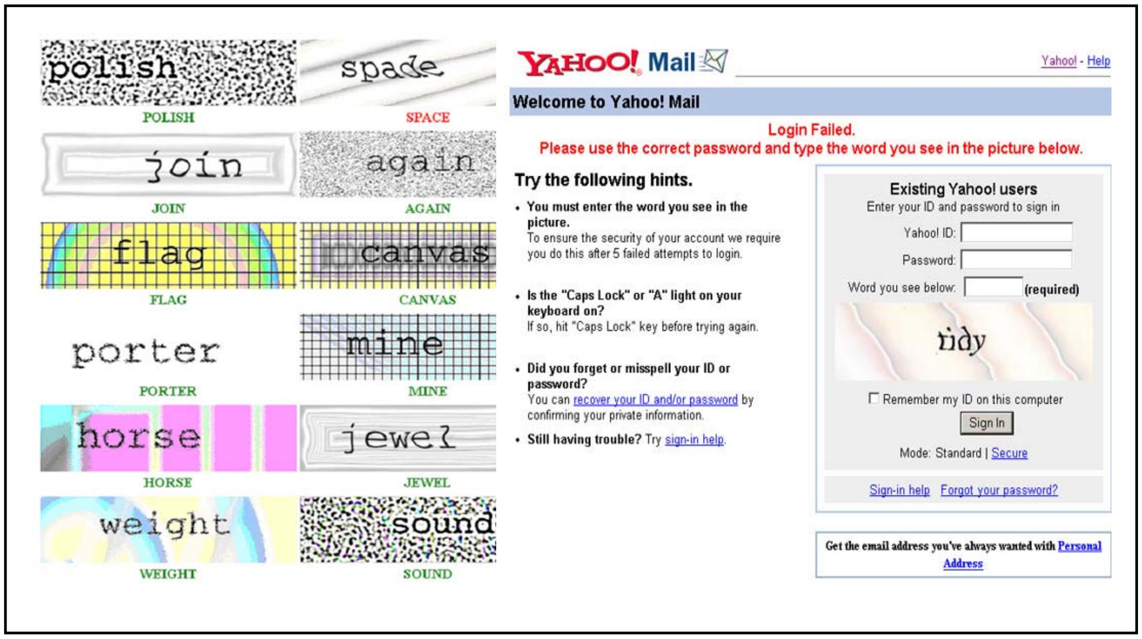
Electronic email systems have evolved over a period of time and the resulting systems face challenges with flexibility, security, and interoperability with other messaging systems. To address these problems there are essentially different choices, such as incremental overlaying, redesigning from scratch, or basing a system on different foundation. The first two approaches have been extensively explored but without fully resolving the issues with flexibility, security, and scalability. The third option has also been explored, but such systems have not been standardized and also face scalability problems. Due to recent developments in standards bodies (W3C [28], OASIS [29] etc), the third option has become viable and WSEmail tries to explore this approach by basing an email system on the emerging infrastructure for web services.

Web services provides a service oriented infrastructure based on data interchange protocols such as SOAP, WSDL, and other XML-based formats developed at W3C and other standards bodies. These protocols aim to provide standardized, secure, flexible and interoperable infrastructure level support for B2B and B2C systems.

WSEmail messages are SOAP messages that use web service features like security and routing to support integrity, authentication, and access control for both end-to-end and hop-by-hop message transmissions. Extensibility in WSEmail is achieved by developing it as a collection of services that can be added or extended to the base system. This also provides a mechanism to integrate different messaging systems such as the usual email SMTP based messages with various kinds of instant messaging systems. The prototype system based on web services illustrates potential for new ideas in security, flexibility, and integration using pluggable applications such as on-demand attachments, routed forms, and integrated instant messaging.

In this case study we have extended the functionality of WSEmail to provide support for adaptation. For this purpose we have developed a puzzle based anti spam functionality for WSEmail. In the perspective of AMPol, we have also modified the WSEmail core implementation to support policy based messaging. These functional extensions to WSEmail have been incorporated without significantly modifying the core implementation. By using this puzzle based anti spam scenario we have tried to show the adaptation in WSEmail messaging system with respect to three adaptation perspectives which we have discussed before in AMPol architecture. This extended WSEmail version (further on WSEmail extended version will be referred as WSEmail-Puzzle) supports the basic implementation of our proposed AMPol architecture. It supports static and dynamic policies along with basic policy negotiation, advertisement and merging. For system extension we have only implemented the Component Extension Manager with limited functionality and extension support.

Puzzle based anti spam solutions already exist [32], [34] and a lot of related work has been done in the area of DOS attacks [35], [33]. Puzzle based anti spam solutions increase the cost of sending email in order to create a computational symmetry between sender and receiver. So the idea is that if you want to send me email messages then you have to perform certain task or solve a puzzle for me and then you can send me a message. Certainly this idea to some extent can avoid spam in our message in-bin. In our WSEmail-Puzzle implementation we have provided support for two types of puzzles: Reverse Turing Test (RTT) and Hash Cash puzzles. RTT puzzles are user interactive puzzles which involves human interaction to solve a graphical problem, as shown in figure 8. Hash Cash are usually non-interactive and computationally bound puzzles. In these types of puzzles, the sender has to allocate memory or processor resources to solve a specific problem (e.g. in case of Hash Cash it is to find hash collisions). Our scope of this study is not to sell the idea of puzzle based anti-spam but to show that how a system can be adapted to support new functionality and this new functionality can be a puzzle-based anti-spam feature in an email system.



YAHOO! Mail

Yahoo! - Help

Welcome to Yahoo! Mail

Login Failed.
Please use the correct password and type the word you see in the picture below.

Try the following hints.

- You must enter the word you see in the picture.
  To ensure the security of your account we require you do this after 5 failed attempts to login.
- Is the "Caps Lock" or "A" light on your keyboard on?
  If so, hit "Caps Lock" key before trying again.
- Did you forget or misspell your ID or password?
  You can recover your ID and/or password by confirming your private information.
- Still having trouble? Try sign-in help.

Existing Yahoo! users
Enter your ID and password to sign in

Yahoo! ID:
Password:
Word you see below: (required)

Remember my ID on this computer

Sign In

Mode: Standard | Secure

Sign-in help Forgot your password?

Get the email address you've always wanted with Personal Address

Fig 8: RTT Puzzle Examples

5.1.2 Architecture and Implementation

WSEmail architecture is based on AMPol Four-Node messaging model. Figure 9 gives a high level architectural overview of WSEmail.

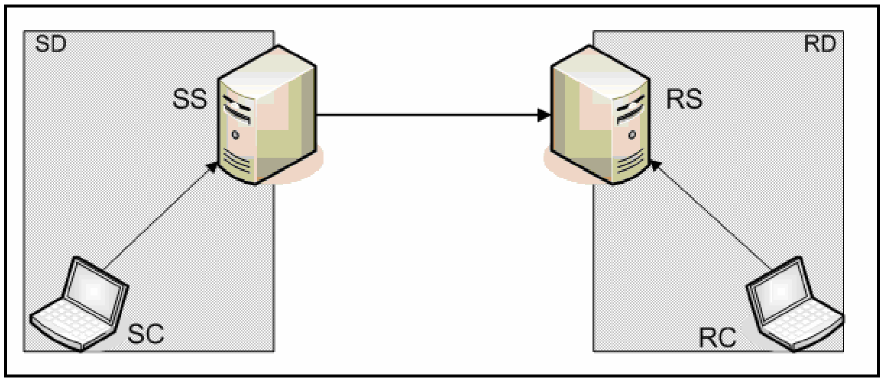


Fig 9: WSEmail Architecture

A Sender Client SC makes a call on its Sender Server SS both residing in the same domain. These calls are SOAP calls over TCP because SS is a web service. The server

SS then makes a SOAP call on the Receiver Server RS to deliver the mail from the Sender Domain SD into the Receiver Domain RD. The Receiver Client RC periodically makes calls to RS to find new messages or download message bodies.

Security is based on standards for web service security, supported by encrypted tunnels. Clients like SC and RC are typically authenticated to their servers by a password, while servers authenticate themselves using certificates. Such certificates are used in TLS and used to sign messages using XMLDSIG. For instance, the message from SC to RC will be given an XMLDSIG signature by SS that is checked by both RS and RC.

5.1.2.1 WSEmail Server Architecture

There are three major entities at the server-side WSEmail system: the core email server, database server and DNS server. The core email server is basically a web server that is responsible for dealing with receiving and distributing mails to both clients and other servers. The database server stores messages for the email server. The DNS server publishes service (SRV) records so that remote WSEmail servers can locate each other. Software components of the WSEmail server are illustrated in Figure 10:

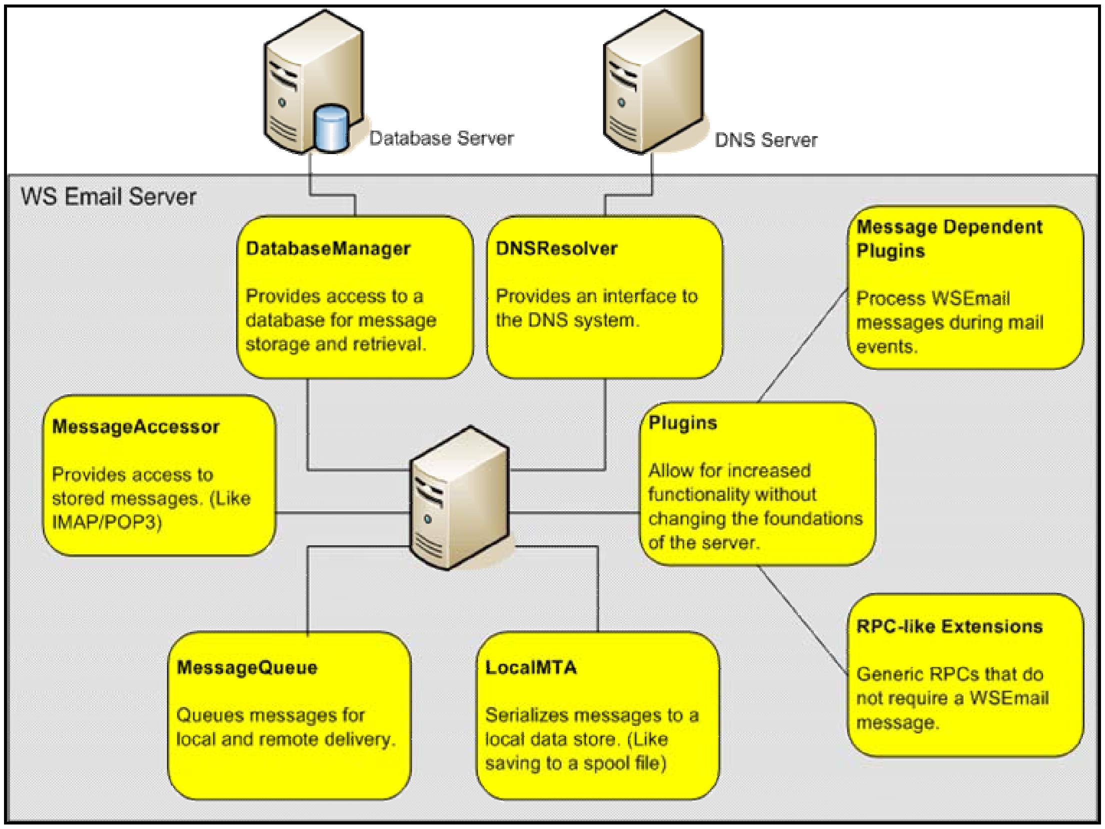

Database Server
DNS Server
WS Email Server
DatabaseManager
Provides access to a database for message storage and retrieval.
DNSResolver
Provides an interface to the DNS system.
Message Dependent Plugins
Process WSEmail messages during mail events.
MessageAccessor
Provides access to stored messages. (Like IMAP/POP3)
Plugins
Allow for increased functionality without changing the foundations of the server.
RPC-like Extensions
Generic RPCs that do not require a WSEmail message.
MessageQueue
Queues messages for local and remote delivery.
LocalMTA
Serializes messages to a local data store. (Like saving to a spool file)

Fig 10: WSEmail Server Architecture

5.1.2.2 WSEmail Plug-In Architecture

The WSEmail server is built upon multiple pluggable interfaces. This fundamental design feature means that any of the major parts of the server are swappable and can be upgraded without changes to other pieces. Similarly it provides easy functionality extension on top of its core implementation. An important feature of WSEmail is its plug-in architecture and its functional extension through these plug-ins. Most critical components are written as plug-ins and major features such as instant messaging and federated identities are simply plugged in, which demonstrates the extensibility of the server and the protocol. This functionality, combined with the client's ability to download and use new programming, enables a new dynamic email system.

There are basically two types of plug-ins, Message-Dependent and RPC-Like. Message dependent plug-ins receive and process a WSEmail messages to perform specific tasks e.g. message filtering etc. The RPC-like plug-ins take XML based arguments as input to perform specific functions not necessarily dependant on an email message.

5.1.2.3 WSEmail-Puzzle Architecture

Our effort aims to provide policy based adaptable messaging system and for this purpose we have to modify the WSEmail system without significantly changing its core implementation. We basically have to achieve three adaptation perspectives, which are:

- policy and requirement adaptation
- adaptive policy/requirement discovery and negotiation
- adaptive policy enforcement and system extension supporting policy enforcement

Now we will discuss which components we have added and modified to achieve these features in WSEmail.

Policy Adaptation and Negotiation:

The basic situation can be seen in the figure below. Here a sender uses his Sender Mail User Agent (SMUA) to send a message to a recipient MUA (RMUA) via a Sender Mail Transfer Agent (SMTA). This message passes through a receiver MTA (RMTA) of the recipient. The recipient has ingress policy for what mail he is willing to receive. There are also policies associated with the relays (SMTA and RMTA): an egress policy for outbound messages and an ingress policy for inbound messages. So a message going from SMUA to RMUA must comply with SMTA egress policy, RMTA and RMUA ingress policies. To comply with these policies, SMUA may have to adapt its functionality and for this purpose it can dynamically extend its functionality by integrating plug-ins from a trusted server.

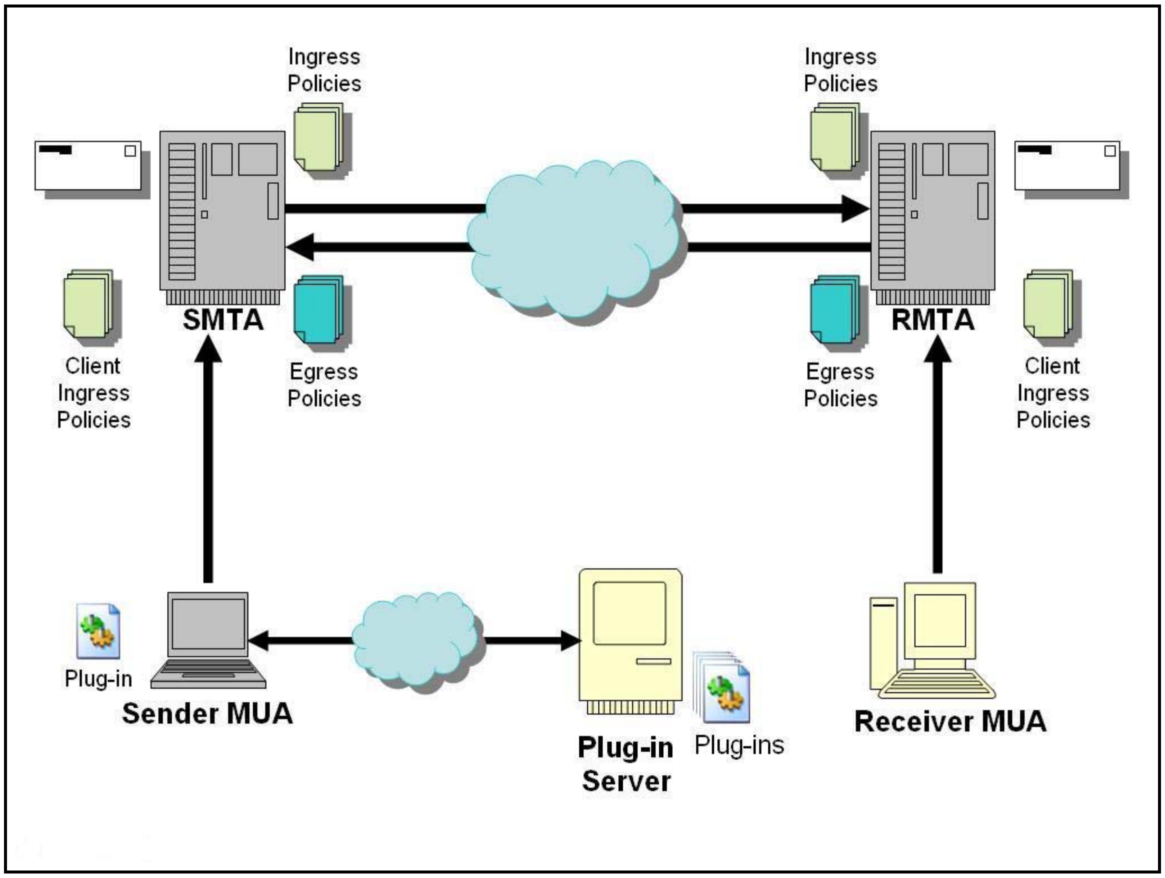


Fig 11: WSEmail-Puzzle Architecture

We have added a ***Policy Manager*** component for policy evaluation and enforcement for each node in WSEmail. We have designed an XML based policy language specifically for WSEmail which is based on APES rules, but only supports ***A*** and ***P*** part of AMPol policy model. We represent messaging requirements and constraints in the form of policy and there are two types of policies in our system: ingress and egress policies. Server nodes (SS/SMTA & RS/RMTA) can have both types of policies and client nodes (SC/SMUA & RC/RMUA) only have ingress policies. In WSEMail-Puzzle architecture, all policies reside at server nodes (SMTA/RMTA), and MUA’s do not manage or store any policies. This approach is useful because as in email systems, server MTA’s are usually online 24x7 hrs and can provide instant access to client policies.

WSEmail-Puzzle also supports dynamic policies, policy query and negotiation. In above scenario, in order to send a message from RMUA to SMUA, SMTA (on behalf of RMUA) queries corresponding RMTA for ingress policies. RMTA merges its ingress policy with RMUA ingress policy and reply back with merged policy. SMTA then evaluates these merged policies and only send related rules merged with its own egress policy to SMUA. SMUA also evaluates the rules of resultant merged policy and accordingly create a message which complies to this merged policy. Accordingly SMTA, if required, perform certain operations on a message and finally sends the message to RMTA, which should now be delivered to corresponding MUA as it satisfy all the corresponding requirements of each node. RMTA first checks the message against its ingress policy and then against RMUA policy. If message evaluation is successful then RMTA puts the message in MUA mail queue to be retrieved by MUA whenever it comes online.

Dynamic policies are generated by flat merging of policies of different nodes. We do not handle conflicts and dynamically merged policy is simply a union of rules. Figure 12 below shows sample static policy for RMTA, which has three sections for Ingress, Egress and Local policy.

```xml
<WSEmailPolicy xmlns:xsd=http://www.w3.org/2001/XMLSchema
       type="static" node="WSEmailMTA">

<Ingresspolicy xmlns="http://securitylab.cs.uiuc.edu/WSEmail">
       <PolicyRules>
              <PolicyRule Id="#05" type="P" DllName="RTT.dll"
              RetrieveURI="http://seclab-london.cs.uiuc.edu/classes/RTT.dll"
              FriendlyName = "RTT No.1" Version = "1.0">
                     <Key>RTT</Key>
                     <Description>RTT Puzzle</Description>
              </PolicyRule>
              <PolicyRule Id="#08" type="A" >
                     <Key>ZIP+PIF</Key>
                     <Description>No Zip and PIF Attachments</Description>
                     <Condition>~(type=*.zip, *.pif)</Condition>
              </PolicyRule>
       </PolicyRules>
</Ingresspolicy>

<Egresspolicy xmlns="http://securitylab.cs.uiuc.edu/WSEmail">
       <PolicyRules>
              <PolicyRule Id="#09" type="A" >
                     <Key>ZIP</Key>
                     <Description>No Zip Attachments</Description>
                     <Condition>~(type=*.zip)</Condition>
              </PolicyRule>
       </PolicyRules>
</Egresspolicy>

<Localpolicy xmlns="http://securitylab.cs.uiuc.edu/WSEmail">
       <PolicyRules>
              <PolicyRule Id="#01" type="A" DllName="HashCash.dll"
              RetrieveURI="http://seclab-
              london.cs.uiuc.edu/classes/HashCash.dll"
              FriendlyName = "HashCash No.1" Version = "1.0">
                     <Key>HashCash</Key>
                     <Description>HashCash Puzzle</Description>
              </PolicyRule>
       </PolicyRules>
</Localpolicy>

</WSEmailPolicy>
```

Fig 12: RMTA Policies

The Local Policy is another type of policy specifically for messages bounded within same domain. The Ingress policy has two rules for inbound messages. It requires sender to solve RTT puzzle and message should not have attachments with zip or pif file extensions. The Egress policy does not allow messages to go out of domain which have attachments of zip files. The Local policy requires sender to solve HashCash puzzle in order to send message to other recipient with in the same domain.

```
<WSEmailPolicy xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       type="static" node="WSEmailMUA">

<Ingresspolicy xmlns="http://securitylab.cs.uiuc.edu/WSEmail">
       <PolicyRules>
              <PolicyRule Id="#01" type="A" DllName="HashCash.dll"
              RetrieveURI="http://seclab-
              london.cs.uiuc.edu/classes/HashCash.dll"
              FriendlyName = "HashCash No.1" Version = "1.0">
                     <Key>HashCash</Key>
                     <Description>HashCash Puzzle</Description>
              </PolicyRule>
              <PolicyRule Id="#09" type="A" Level="Private" >
                     <Key>Size</Key>
                     <Description>Small Attachments</Description>
                     <Condition>(size<=20KB)</Condition>
              </PolicyRule>

       </PolicyRules>
</Ingresspolicy>

</WSEmailPolicy>
```

Fig 13: RMUA Policies

Similarly Fig 13 and 14 show the static policies of RMUA and SMTA. The final merged dynamic policy at SMTA is shown in figure 15, which is just a union of corresponding rules in the Egress and Ingress policies. The merged policy will be send to SMUA, which then has to construct a message complying to all defined rules in the policy.

```
<WSEmailPolicy xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      type="static" node="WSEmailMTA">

<Ingresspolicy xmlns="http://securitylab.cs.uiuc.edu/WSEmail">
</Ingresspolicy>

<Egresspolicy xmlns="http://securitylab.cs.uiuc.edu/WSEmail">
      <PolicyRules>
            <PolicyRule Id="#09" type="A" >
                  <Key>ZIP</Key>
                  <Description>No Attachments</Description>
                  <Condition>~(type=*.*)</Condition>
            </PolicyRule>
      </PolicyRules>
</Egresspolicy>

<Localpolicy xmlns="http://securitylab.cs.uiuc.edu/WSEmail">
</Localpolicy>

</WSEmailPolicy>
```

Fig 14: SMTA Policies

In the WSEmail-Puzzle, every server MTA provides a publically accessible interface to clients for acquiring merged dynamic policies. As in WSEmail, the MTA server is based on web service, so we have provided a web method for acquiring dynamic policies for a particular recipient. MTA's do not publish static policies and only provide access to dynamic policies based on client credentials, which provides a secure and controlled mechanism to publish MTA policy rules. These dynamically policies are usually subset of static policies and only contain those rules which are not considered confidential. For example the second rule in RMUA policy is confidential and it is not included in the merged policies.

```
<WSEmailPolicy xmlns:xsd=http://www.w3.org/2001/XMLSchema
       type="merged" node="WSEmailMUA">

<Ingresspolicy xmlns="http://securitylab.cs.uiuc.edu/WSEmail">
       <PolicyRules>
              <PolicyRule Id="#01" type="P" DllName="RTT.dll"
              RetrieveURI="http://seclab-london.cs.uiuc.edu/classes/RTT.dll"
              FriendlyName = "RTT No.1" Version = "1.0">
                     <Key>RTT</Key>
                     <Description>RTT Puzzle</Description>
              </PolicyRule>
              <PolicyRule Id="#02" type="A" >
                     <Key>ZIP+PIF</Key>
                     <Description>No Zip and PIF Attachments</Description>
                     <Condition>~(type=*.zip, *.pif)</Condition>
              </PolicyRule>
              <PolicyRule Id="#03" type="A" DllName="HashCash.dll"
              RetrieveURI="http://seclab-
              london.cs.uiuc.edu/classes/HashCash.dll"
              FriendlyName = "HashCash No.1" Version = "1.0">
                     <Key>HashCash</Key>
                     <Description>HashCash Puzzle</Description>
              </PolicyRule>
       </PolicyRules>
</Ingresspolicy>

<Egresspolicy xmlns="http://securitylab.cs.uiuc.edu/WSEmail">
       <PolicyRules>
              <PolicyRule Id="#04" type="A" >
                     <Key>NO</Key>
                     <Description>No Attachments</Description>
                     <Condition>~(type=*.*)</Condition>
              </PolicyRule>
       </PolicyRules>
</Egresspolicy>

</WSEmailPolicy>
```

Fig 15: Merged Policies

To provide the features related to Policies and Negotiation we have added a Policy Manager Component to WSEmail which has following modules:

- **Policy Decision Module:** It verifies the message whether it comply with specific policy or not.
- **Policy Loader:** It loads ingress and egress policies during initialization phase. It also loads egress policies of SC/SMUA on request.

- **Policy Merger Module:** It merges two policies on the basis of flat merging rule which is simple a union of rules of corresponding policies. It discards confidential rules in the merging process.
- **Policy Publisher:** It provides a SOAP interface for querying ingress policies. On receiving a request it calls the Policy Merger Module to merge RMTA and RMUA ingress policies and returns a merged policy back to requestor.
- **Policy Evaluator:** This module controls the whole policy evaluation and enforcement process. It asks the Policy Decision Module to check the compliance of the message with specific policy. It uses Policy Loader to load policies and make them available to Policy Publisher and Merger Modules.

Figure 16 below shows above discussed modules and components for WSEmail-Puzzle Server (MTA's).

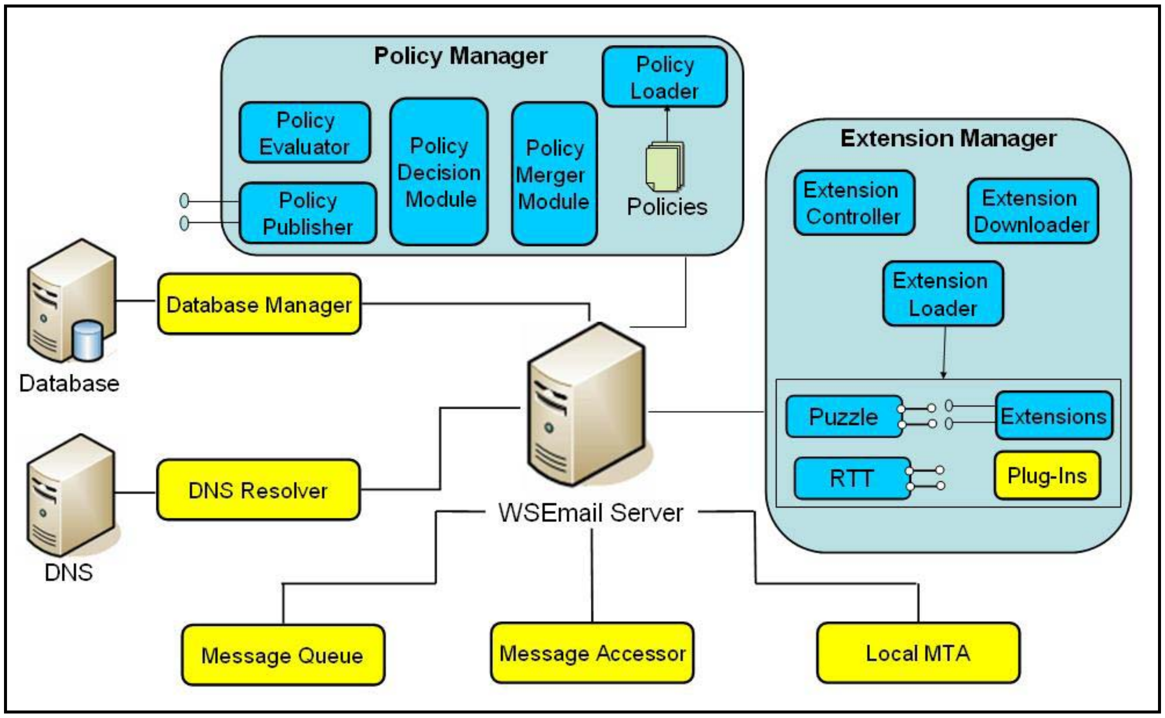


Fig 16: Modified WSEmail Server Architecture

Similarly WSEmail Client (MUA's) also has the same Policy Manger Component for evaluating and enforcing policies. In this Policy Manager we do not need Policy Publisher and Policy Merger Modules as we do not need to publish any MUA's policies.

Policy Enforcement and System Extension

Every WSEmail node (Server MTA and Client MUA) has a Policy Manager Component to evaluate and enforce their corresponding policies. If for some rule, a particular node does not have any capability to enforce or satisfy the requirements specified in that rule, then that node may need to download a plug-in or an extension, which can be used to satisfy the requirements. For this purpose we can also use already built in support for WSEmail plug-ins, but these plug-ins are specifically built for WSEmail and they only work if WSEmail plug-in framework is in place. In our implementation we have modified the WSEmail plug-in architecture to make it more generic and dynamically extensible. For this purpose we have modified the plug-in architecture in accordance with our AMPol Extension model and implemented an Extension Manager which also supports the WSEmail specific plug-ins along with other generic plug-ins and extensions. The Extension Manger is integrated in both WSEmail Server and Client, thus providing extensibility throughout whole WSEmail System. Although this Extension Manger Component is a simplified version of what we have proposed in AMPol Extension Model, but it serves our purpose here regarding dynamic and generic extension of WSEmail system.

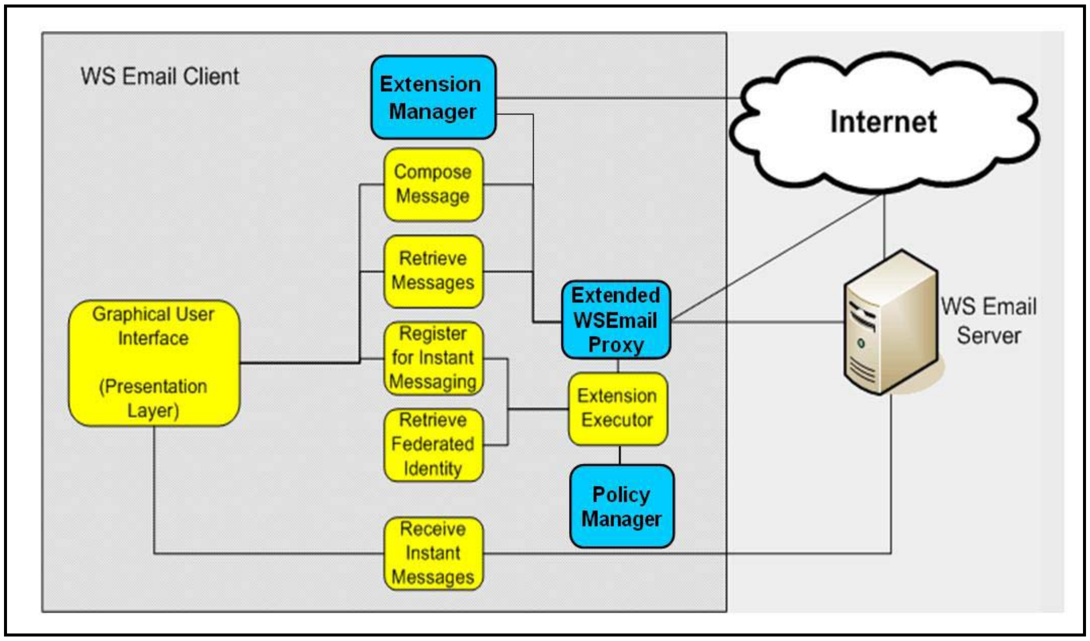


Fig 17: Modified WSEmail Client Architecture

5.1.2.4 Implementation

WSEmail-Puzzle prototype runs on Windows server and client systems. Our implementation and modifications are implemented over the .NET framework and relies on Web Services Enhancement (WSE) 1.0, CAPICOM 2.00, SQL Server 2000 (to store messages for the server), and IIS 5.0. Currently Extension Manager only provides support for Windows based DLL plug-ins or COM based components. These plug-ins do not need to comply with the WSEmail Plug-in architecture as it was in previous implementation of WSEmail.

5.1.3 Discussion

In above discussion we have shown modification to WSEmail for simplified implementation of AMPol Policy, Negotiation and Extension Model. WSEmail-Puzzle supports Static and Dynamic policies based on AMPol APES rules. It also supports policy advertisement and simple one step policy negotiation for communicating merged policies with clients. Similarly we have achieved a third adaptation perspective of AMPol in WSEmail-Puzzle through incorporation of Extension Manager.

We have tried to address the problems of flexibility, security, and interoperability faced to current messaging systems by basing our system on entirely new foundation. The WSEmail-Puzzle based on AMPol, tries to explore this approach by basing an email system on the emerging infrastructure for web services. Web services inherently provide flexibility and interoperability, and WSEmail-Puzzle exploit this feature to achieve a highly adaptable email system. For security purposes, the WSEmail-Puzzle uses Web Services Enhancement (WSE) security features, which provide secure data transfer on top of web services protocols.

Although what we have shown is a minimal implementation of proposed AMPol models, but a simple working prototype successfully shows an implementation proof of our concept.

5.2 Adaptive VoIP Call System

5.2.1 Introduction

This case study discusses the architecture and development of extensible call management system, called Adaptive Calling System (ACS), which supports connection and call management between VoIP clients and PSTN telephone clients.

ACS helps VoIP clients to dynamically establish a connection between each other. Our aim is not to build a complete call management system with rich functionalities, but to build an adaptable system with basic functionalities emphasizing on adaptation feature.

ACS assists in establishing connection and managing calls for client communicating over:

- VOIP to VOIP
- VOIP to PSTN Telephone lines

ACS system is based on the AMPol model which manages connections between two VOIP clients and VOIP & Telephone clients. Each client specifies its requirements for communication protocols in the form of policy. These requirements are negotiated with each other and both clients agree upon it in order to establish a successful connection. If a particular client does not support a required protocol then he can dynamically extend its functionality to add a component which can communicate with the other client using required protocol.

We will try to show the extensible nature of the ACS by dynamically extending the functionality of its clients to use multiple VoIP protocols per demand.

5.2.2 Background

Voice over IP (VoIP) uses the Internet Protocol (IP) to transmit voice as packets over an IP network. Using VoIP protocols, voice communications can be achieved on any IP network regardless it is Internet, Intranets or Local Area Networks (LAN). In a VoIP enabled network, the voice signal is digitized, compressed and converted to IP packets and then transmitted over the IP network. VoIP signaling protocols are used to set up and tear down calls, carry information required to locate users and negotiate capabilities.

The key benefits of Internet telephony (voice over IP) are the very low cost, the integration of data, voice and video on one network and simplified management of end user and terminals.

There are a few VoIP protocol stacks which are derived from various standard bodies and vendors, namely H.323, SIP, MEGACO and MGCP. These are control (signaling) protocols and they create a VoIP connection between clients. Currently ACS supports clients with H.323 and SIP protocol only for P2P VoIP communication.

H.323 is the ITU-T's standard, which was originally developed for multimedia conferencing on LANs, but was later extended to cover Voice over IP. The standard encompasses both point to point communications and multipoint conferences.

Session Initiation Protocol (SIP) is the IETF's standard for establishing VoIP connections. SIP is an application layer control protocol for creating, modifying and terminating sessions with one or more participants. The architecture of SIP is similar to that of HTTP (client-server protocol). Requests are generated by the client and sent to the server. The server processes the requests and then sends a response to the client. A request and the responses for that request make a transaction.

ACS clients currently use the RTP protocol for actual data (voice media) transfer and either SIP or H.323 protocol for control signaling in order to create a connection. Where real-time transport protocol (RTP) provides end-to-end delivery services for data with real-time characteristics, such as interactive audio and video or simulation data, over multicast or unicast network services.

Applications typically run RTP on top of UDP to make use of its multiplexing and checksum services; both protocols contribute parts of the transport protocol functionality.

However, RTP may be used with other suitable underlying network or transport protocols. RTP supports data transfer to multiple destinations using multicast distribution if provided by the underlying network. RTP itself does not provide any mechanism to ensure timely delivery or provide other quality-of-service guarantees, but relies on lower-layer services to do so. It does not guarantee delivery or prevent out-of-order delivery, nor does it assume that the underlying network is reliable and delivers packets in sequence.

In the past few years, the VOIP industry has been working on addressing the many key issues e.g. interoperability, quality of service, scalability, security etc and in ACS we have tried to address few of them by basing our system on AMPol architecture:

Interoperability: In a public network environment, products from different vendors need to operate with each other for voice over IP.

Security: Encryption (such as SSL) and tunneling (L2TP) technologies are developed to protect VOIP signaling and bear traffic, but these techniques are not standardized for VoIP and thus results in highly inflexible and complex system with customized security solutions.

Extensibility: Due to ever-changing requirements especially in the area of protocols and data streaming functionality, VoIP systems must be extensible to support these new requirements. Currently most VoIP systems provide static set of protocols with fix functionality.

Policy Enforcement: As in messaging systems, each client wants to control the incoming and outgoing data flow. Similarly VoIP systems also require policy enforcement mechanism to efficiently control its data communication flow.

Integration with Public Switched Telephone Network (PSTN): While Internet telephony is being introduced, it will need to work in conjunction with PSTN in the foreseeable future. Gateway technologies are developed to bridge the two networks, but communication with gateways requires specific non standardized customization at client end.

5.2.3 Architecture and Implementation

ACS is an extensible and flexible system for P2P VoIP call management. ACS concentrates on System Extension perspective of AMPol and incorporates a simple extension model and a minimal functionality for Policy Management.

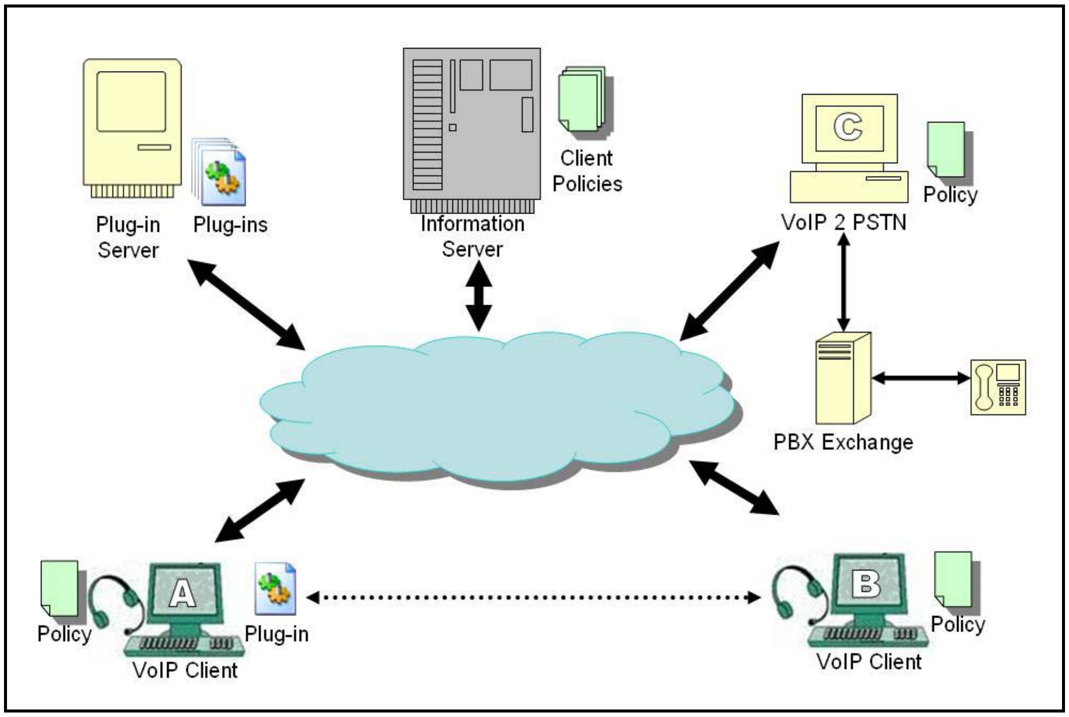


Fig 18: ACS Architecture

Figure 18 above shows an overview of ACS architecture. Every VoIP client in ACS defines its communication requirement and constraints in the form of policy called

Communication Policy. Then it uploads these policies to ***Information Server*** (Policy Server). The Information server serves as a trusted policy publisher, providing online access to policies of different clients. ACS does not support complex policy advertisement and negotiation, instead uses a centralized trusted policy repository to publish and access policies. When any client wants to create a VoIP connection to another client, it first searches for other client policy requirements from Information server and then accordingly initiates a call session.

If it is not able to fulfill the communication requirements of the other client then it either tries to download the plug-in component or discard the communication. Plug-in components are downloaded from a trusted party hosting those components and clients policies have to specify from which third party server these components can be downloaded.

Let's suppose Client ***A*** is only capable of supporting SIP/RTP based VoIP communication and Client ***B*** is capable of communication using H323/RIP protocol. Now if ***A*** wants to create a communication link with client ***B***, A first downloads B's policy from information server and then ***A*** will evaluate it. ***A*** will find that in order to communicate with ***B*** it must support H323/RIP protocol, and it can also find from policy that there is a third party component available which can be used to establish a successful link with ***B***. If policy has this meta-information for downloading component then ***A*** will download and use that component to create a H323/RIP based communication session with ***B***.

Similarly suppose there is another client ***C*** which provides VoIP to PSTN services i.e. it can convert VoIP traffic to PSTN telephone traffic and can connect VoIP clients to telephone lines. Here ***C*** is working like a media gateway between VoIP and PSTN links. Say for this purpose it supports a customized protocol for signaling and RTP for voice media (e.g. C/RTP) then for ***A***, to connect with ***C*** and to use its services, has to download a required component which support C/RTP protocol. This customized protocol requires extra control information such as target telephone number etc, and for this, the downloaded plug-in component can popup a GUI screen and ask the user for extra information.

In the Figure below we have shown some related components in server and client for achieving above discussed behavior. The server does not need any extensible behavior, it only requires components for uploading, storing, searching and publishing policies. Clients require a ***Policy Evaluation*** component for evaluating the requirement of other clients. It has a ***Policy Decision Module*** to verify that the incoming connection request conform to its communication policy or not. The ***Policy Downloader*** simply searches for a communication policy of the target client with which it wants to establish a communication channel. In ACS, for real time performance, communication policies are only for signaling (establishing connection) phase and so policy verification is only done during channel creation and call setup. For media transfer there are usually no policy constraints, because for secure and reliable communication, the security and reliability requirements are only required during call set-up. During this call set-up, secure and reliable channel are created for media transfer e.g. two parties can establish a secure connection for VoIP over SSL link and can achieve reliability by RTP over TCP. Along with protocol types, security and reliability requirements can be specified in communication policy, while APES rules of AMPol only concentrate on security, puzzle and attachment related rules.

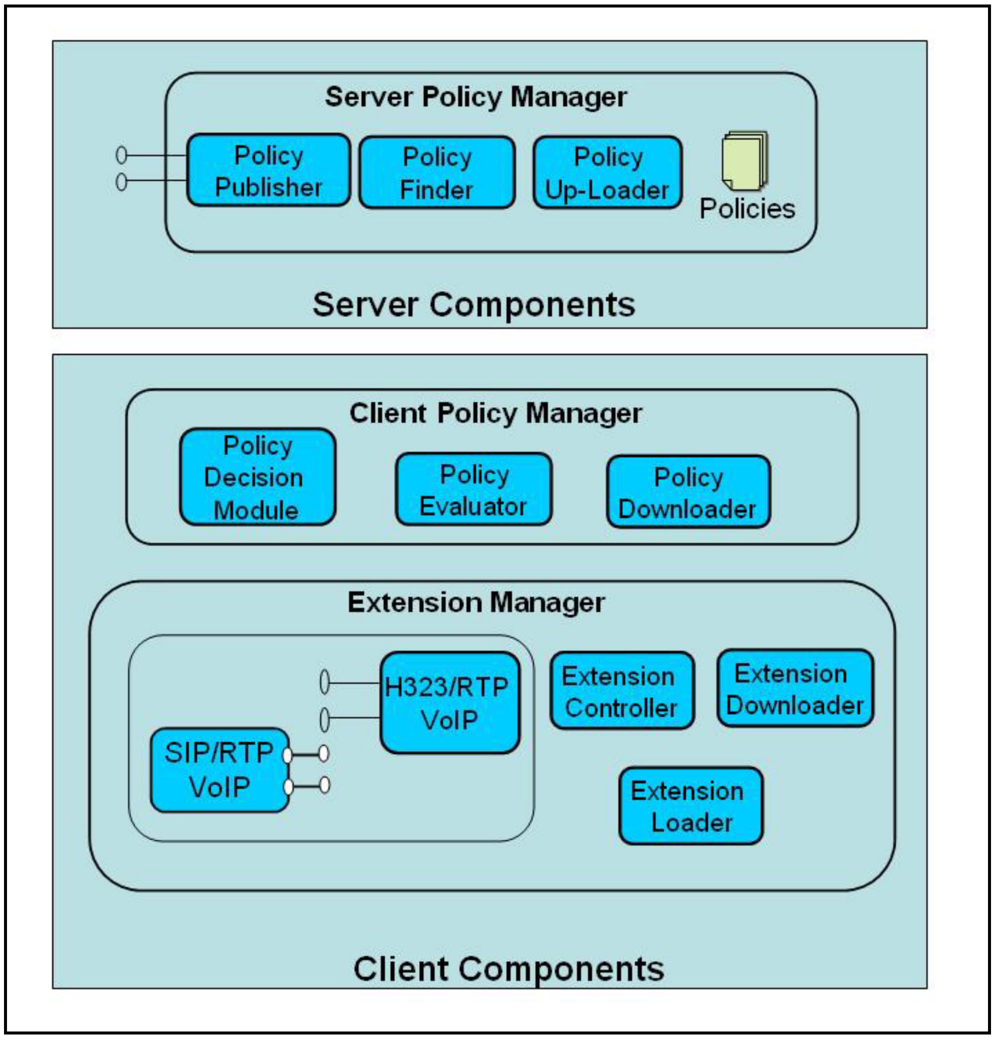


Fig 19: ACS Components

Clients have an ***Extension Manager*** to download and integrate plug-ins from the trusted third party. The ACS Extension Manager has similar functionality and properties as of AMPol Extension Manager.

We have implemented ACS system in SmallTalk using SQUEK environment. SIP/RTP and H323/RTP plug-in components are developed in C++ as windows DLL. While a client having VoIP to PSTN functionality was developed in Java 1.2+ and for that purpose we have also implemented Policy and Extension Manger in Java. Now we are working toward a demo showing above discussed scenario for client ***A***, ***B*** and ***C***.

5.2.4 Discussion

In this study we have discussed architecture and development of ACS for simplified implementation of AMPol Policy and Extension Model. ACS supports Static policies extended from AMPol APES rules. It supports policy advertisement through dedicated Policy Server. System extension is achieved through incorporation of Extension Manager at clients. Although what we have shown is a minimal and modified implementation of proposed AMPol models, but it successfully shows how declarative requirements in the form of policy can help system achieve adaptability and flexibility.

CHAPTER 6: RELATED AND FUTURE WORK

6.1 Introduction

This section presents a detail overview of comparison of our work with other related research efforts. In the end, future plans are also discussed which gives a general overview of our future milestones to achieve for AMPol project. We have identified three related research areas based on our three adaptive perspectives and tried to point out some commonalities and differences.

6.2 Policies and Negotiation

There has been lot work done on Policy Models, Languages and Negotiation but here we will limit our discussion to Policy Negotiation in ATN, Policy Management, Policy Composition and Policy Based Email Systems. These areas have direct relation with the AMPol model and perspectives, and we will analyze them with adaptation perspective in mind.

6.2.1 Trust Negotiation and Policy Management

The AMPol Model shares a lot of properties and problems encountered in research area of Automated Trust Negotiation [23], [24], [25], [26] (ATN). ATN is a new approach to access control and authentication in open, flexible systems. It enables open computing by assigning an access control policy to each resource that is to be made accessible to outsiders. An access control policy describes the properties of the parties allowed to access that resource, in contrast to the traditional approach of listing their identities. An attempt to access the resource triggers a trust negotiation, in which the resource owner and the requesting party take turns disclosing relevant access control policies and digital credentials to one another. In AMPol and WSEmail, a particular client messaging resources have the same meanings as of a resource in ATN. Similarly as in ATN, AMPol also specifies policies which describe requirements and constraints to access these resources for a successful communication. In contrast AMPol emphasize on adaptation aspects of a system rather than trust.

The AMPol and the ATN approaches heavily rely on policies and hence both face issues and problem related to policy management. The policy management is a relatively new field of endeavor, of which the Ponder project and KAoS's KPAT are the most prominent and most advanced examples [19], [20], [21], [22]. The policy management research has not yet addressed the related challenges of large-scale policy deployment, such as ways to help users understand large numbers of complex, interrelated policies, so that they can quickly and effectively change them in response to evolving needs. The policy management issues alone could be overwhelming to solve and there are few areas in ATN which overlaps with policy management for AMPol. These areas can be policy configuration management, algorithms for analysis of policies for consistency, evaluation tools and test beds for testing policy conformance etc.

In the trust negotiation, negotiating parties usually rely on policy disclosures to learn each other's access control requirements and these policies may contain sensitive information. This problem can be avoided by using meta-policies: policies for disclosing other policies. Even with this protection, an adversary may be able to infer private information by observing behavior of a negotiator. AMPol policy disclosure and negotiation mechanism also faces this problem.

In this perspective, approaches used in ATN can also be used for AMPol policy disclosure and negotiation, such as:

- An improved understanding of the application domain factors that may prevent the sharing of resources and an understanding of the kinds of policies for sharing information between unknown entities.
- Policy analysis algorithms and management tools that allow users to easily create, understand, and update a large set of policies, and to control who is allowed to make changes under different circumstances.
- An investigation of radically different cryptographic approaches for disclosing and migrating credentials or policies that will provide improved resilience against attacks against privacy.
- A design and implementation of modular, reusable, scalable, and portable components for policy management and negotiation strategies.

6.2.2 Policy Composition

Efficient algorithms are required to solve issues like how to merge policies, especially policies for different domains. New techniques like defeasible policy merging [15] can be applied for AMPol policy merging which can provide an efficient and readable priority based policy merging mechanism.

Lee [15] et al. in their work, proposed a solution to the problem of composing WS-SecurityPolicies. They have presented an abstract framework for specifying policies in semantically enhanced way and a mechanism for easy merging and composition of policies with other policies in an automated fashion. They also presented an instance of this framework based on defeasible logic. Defeasible logic very closely models the ways in which human's reason and solves problems, making it excellent candidate logic for the specification of security polices and composition rules. Enhanced policies described using defeasible logic can change the ways in which they are composed with other policies based upon conclusions drawn by other parties in the coalition or by examining high-level context information regarding the environment in which the web service is to be deployed.

6.2.3 Policy Based Email System

Kaushik et al. proposed a policy driven approach for email systems in order to tackle spam problem. In their work they tried to overhaul the email so that it can continue to serve as the valuable resource that it has been in the past. There work is quite similar to policy driven approach used in WSEmail-Puzzle. Both solutions view a mailbox as a network resource of the recipient and email messages as access requests from senders, requesting write access to a mailbox. In their approach, they replace the current notion that an arbitrary user has an implicit right to place a message in anyone's mailbox with a scalable, attribute-based access control policy in which a sender's access to a recipient's mailbox is controlled by the recipient. Furthermore in order to grant access rights to sender they have also used the notion of sender authentication and gradual trust establishment (trust negotiation) along with reputation of each entity in the email system.

6.2.4 Petmail

The Petmail [31] (P.E.T: Permission, Encryption and Transparency) is a Permission-based Anti-Spam System. Basic idea of Permission-based Anti-Spam is to consume some of my time without explicit permission; you must be willing to give up some of your own. A mailbox is a kind of resource of recipients. Besides the storage that is required to store the message, recipient needs to spend some time to deal with incoming messages. Messages also costs bandwidth of a mail server and all these parameters (time, storage, and bandwidth) are a resource. If someone wants to access these resources, it must get some permission. In WSEmail-Puzzle and generally in AMPol, we have the same idea of policy based access to target client's messaging resources.

The Petmail uses a novel data structure called Identity Record to store the information of each side of email system. It proposed an Address Server to accomplish the name search, look up the permission (policy) and routing information. Message transportation is accomplished by the Transport Provider and Address Server. It applies some anonymous transport technologies to relay messages in order to achieve message sender and receiver privacy. It uses an Agent Component to communicate with the rest of the world on user's behalf. Agent enforces the user's policies, and implements the Petmail protocol. Ticket Server is used to implement CAPTCHA challenges, a type of puzzles. The Ticket Server produces signed Ticket, which is a statement that some given sender has passed some challenge. This Ticket is labeled with the Ticket Server's URL or other identifying string, and then included in a Petmail message to make it eligible for acceptance by a given recipient.

If a sender never sent a message to recipient before, according to recipient's IDRecord, the recipient's Petmail agent only grants generic permissions to the sender, i.e. redirects the sender to Ticket Server. Sender has to answer a CAPTCHA or Hash Cash challenge which cost some resources of sender. If the solution is correct, Ticket Server issues a ticket to sender. Together with the message body, the ticket is sent the recipient’s agent. The agent verifies ticket, signature, etc. If the verification succeeds, the message is delivered to recipient and some specific permission is granted to sender. Generally, sender does not need to apply ticket next time.

In the AMPol WSEmail-Puzzle, MTA not only enforces the policies but also proposes and verifies the Puzzle solution. While in Petmail, verification of the puzzle solution is accomplished by the third party, Ticket Server. The Petmail approach is relatively more flexible and maintainable for this application. It is more flexible because if recipient does not trust Ticket Server anymore, it can subscribe to any other Ticket Server. In the WSEmail-Puzzle, MUA only can take advantage of MTA’s Puzzle components. But WSEmail-Puzzle can easily extend its policy verification mechanism specifically for puzzles and delegate it to third party. In order to achieve this we just have to replace current policy verification module with the modified one, which support remote puzzle verification. This approach is more maintainable because if puzzle technology is improved and we need to update current puzzle implementation then for the WSEmail-Puzzle we have to integrate new puzzle components. In Petmail, it is outsourced to Ticket Server and only Ticket server needs to be updated.

Petmail introduces a simple Trust Model which is not explicitly addressed in WSEmail-Puzzle and AMPol. It is practical in application that if a user has satisfied some requirement, he is not willing to repeat the same steps to set up the secure connection with the same correspondent again. It is tedious and waste of resources.

In the Petmail, the policy (permission) is pre-defined by recipients. The policy (permission) is static and there is no negotiation process. AMPol generates dynamic policies and two sides can exchange information with each other. While in Petmail, the permissions are fixed and they can not change dynamically based on environment or context.

The Petmail assumes that if sender wants to send an email to the recipient, sender trusts the recipient and grants specific permission to recipient, i.e. allows recipient to send back email to sender without a ticket. It is possible that the spammer has the victim send a message to spammer at first, then the spammer gains the access to victim's mailbox immediately.

In Petmail, all messages are required to signed and encrypted. The recipient needs to do computation for verification and decryption. It is subject to DoS attack if the spammer sends message no matter it gets tickets from Ticket Server or not. Recipient's agent will spend a great deal of resources to deal with these messages. Petmail can not resist such attack efficiently.

6.3 Adaptation and Extension

Adaptation and extensibility of the software systems has been studied by various researchers in different perspectives. Some of the well known efforts are reflection [6], open implementation [7], adaptive programming [8], aspect-oriented programming [9], component composition [10], [14], and collaboration based design [11], as well as quality of service (QoS) policy management in the networking community [12]. In these approaches, there is an entity representing a system execution policy. For example, it is called meta-object in the context of reflection, concerns in the principle of separation of concern [13], component or plug-in in component composition, and aspect in aspect-oriented programming. Applications can adapt to a given requirement by adding, customizing or replacing these entities. Our adaptation framework is also based on extensions and components, and extensibility is achieved by adding, customizing or replacing these entities. Our work is novel because along with dynamic component composition it also supports client-specific customization and the adaptation process itself is adaptive, thus achieving extensibility from perspectives of clients, systems and process itself.

Eddy et al. [2], [3], [4] has proposed a software composition model that supports client-specific and dynamic customization of distributed systems and services. The software composition model, named Lasagne, supports a selective and dynamic injection of crosscutting features into a minimal core system on a per client-request basis. This Lasagne approach on one hand integrates collaboration-based design and component-based development into a unified software decomposition strategy. On the other hand, it provides a technique for composing these components in a dynamic and context-sensitive manner: components are dynamically selected depending on the needs of the context and are incrementally composed into the distributed application at runtime. Lasagne model fulfills most of the requirements which we have outlined before, but there core model lack meta-adaptive features and their model is not policy driven so it has relatively static composition and extensibility logic.

Tan et al. [5] has used a meta-level architecture for managing and adapting policy-based security services. The have define a meta-level system to be an architecture which separates domain knowledge from control knowledge. According to them, there are number of advantages of this type of architecture: it supports a loosely coupled approach to designing and managing knowledge and enables the sharing of knowledge across different systems. This type of meta-architecture enables them detect, analyze and resolve multiple policy conflicts, decide if a change in the environment necessitates a security reconfiguration and to decide if a suitable level of security interoperability between heterogeneous systems is achievable. The AMPol Extension model also separates the domain knowledge from the controlling knowledge. The core Component and Extension Manager Components controls the domain components irrespective of their type and kind. The AMPol Extension model can be used for any application domain components and it is also verified to some extent in our case studies. Furthermore, the AMPol Extension model is also policy driven as the meta-level architecture proposed by Tan et al.

6.4 Future Work

This work only investigates some architectural issues for flexibility and integration in order to develop adaptive policy based system. We have emphasized on the architectural aspects of the AMPol System and proposed architectural models for Policy Adaptation, Policy Negotiation and System Extension. This is just a start toward a comprehensive and generic adaptable messaging system, and many areas need to be thoroughly explored and investigated. Related areas such as formal theoretical foundations, algorithms, interoperability, security, scalability, testing etc. specifically need to be investigated in the context of AMPol.

Efficient algorithms are needed to for merging, evaluating and verifying policies. Policy negotiation protocol verification techniques are needed to compensate for the increased flexibility of the system, which leaves the design of potentially tricky security protocols in the hands of non-specialists. For this purpose we can benefit from advances in automated verification of the correctness of security protocols e.g. systems like TulaFale [9] and ProVerif [10] to provide verifications for web service security protocols.

We also need to explore issues related to interoperability of our proposed system so that it can work with the existing base and to determine the features that offer the most likely practical benefits.

We have not fully evaluated and tested AMPol models, so we need implementation test beds to thoroughly evaluate and analyze them in order to justify our claims. Similarly, for wide scale adoption and to illustrate the effectiveness of our proposed approach we need to develop requirements for applications in different specific areas. In this work we have only presented applications in the area of internet email and VoIP communication to support our proposed theory, but we need to explore other related application domains in order to extend our models and this will also help us determine the features that offer the most likely practical benefits.

In this work, to avoid complexity, we have based AMPol policy model only on APES action rules. In future we plan to propose a generic policy model for AMPol, which can support action rules to model requirements of any application domain.

Similarly there are lots of open questions regarding AMPol Extension Model which need to be addressed. Currently the model only supports stateless client-specific extensions and their customizations. Further work is required to manage complex state-full per session or multi session client-specific dynamic extensibility and customization. In current scope we have limited our work to components/extensions for which there is no or minimal effect on other components and their interaction. In future we need to look into scenarios for maintaining consistency incase of complex system-wide interactions and dependencies of extensions. Also we have not addressed security vulnerabilities and threats to our proposed extension model. We also need to provide a threat model along with possible security solutions to tackle different types of attacks. Similarly in future we plan to critically evaluate the overall performance and reliability of our framework and will propose some enhancements for better efficiency and performance.

CHAPTER 7: CONCLUSIONS

In this work we have explored the idea of developing an adaptive policy based messaging system and tried to solve the limitation faced by the conventional messaging systems. The AMPol approach is to enable parties to advertise communication policies to which a sender can adapt in order to get a message to a recipient on terms acceptable to the recipient. The communicating parties check their own environment, determine and set up dynamic conversation policies through policy advertisement, negotiation and migration. By enforcing the policy, both sides perform agreed operations on the messages, install and execute designated third-party plug-ins and finally send messages between themselves. Thus, this approach present a more flexible and extensible messaging platform to the applications.

AMPol is based on three adaptation perspectives: dynamic adaptation of policies, adaptation of system interaction protocols and adaptation of core application structure. In AMPol, for these three perspectives we have proposed models for Policy Adaptation, Policy Negotiation and System Extensibility. By using these models, we have shown that the system can generate and enforce dynamic policies to react to ever-changing requirements in an adaptive manner. In this work different case studies are described and we have shown that how declarative requirements in the form of policies can help system achieve adaptability and flexibility. In this work we have emphasized on the architectural aspects of the AMPol System and have only presented applications in the area of internet email and VoIP communication to support our proposed theory. In future we plan to investigate other related areas building on our current work.

REFERENCES

[1] RFC 2476 Message Submission.

[2] Eddy Truyen, Bart Vanhaute, Wouter Joosen, Pierre Verbaeten, Bo Noerregaard Joergensen, "Dynamic and Selective Combination of Extensions in Component-based Applications", in Proceedings of the 23rd International Conference on Software Engineering (ICSE'2001), May 2001, Toronto, Canada.

[3] Eddy Truyen, Bart Vanhaute, Wouter Joosen, Pierre Verbaeten, Bo Noerregaard Joergensen, "A Dynamic Customization Model for Distributed Component-Based Applications, accepted for International Workshop on Dynamic and Distributed Multiservice Architectures (DDMA 2001).

[4] Eddy Truyen, Dynamic and Context-Sensitive Composition in Distributed Systems, PhD thesis, K.U.Leuven, Belgium, November 2004

[5] Juan Jim Tan, Stefan Poslad, Yanmin Xi: Policy Driven Systems for Dynamic Security Reconfiguration. AAMAS 2004: 1274-1275

[6] G. Kiczales et al. *The Art of the Metaobject Protocol*. MIT Press, 1991.

[7] G. Kiczales, Beyond the Black Box: Open Implementation, IEEE Software, vol. 13, no. 1, 1996.

[8] K. J. Lieberherr, The Art of Growing Adaptive Object-Oriented Software, PWS Publishing Company, 1995.

[9] G. Kiczales et al., Aspect-Oriented Programming, In Proceedings of the European Conference on Object-Oriented Programming (ECOOP'97), Springer LNCS 1241, 1997.

[10] M. Aksit et al., Abstracting Object-Interactions Using Composition-Filters, In Object-based Distributed Processing, R. Guerraoui et al. (eds.), 1993.

[11] M. Mezini and K. Lieberherr, Adaptive Plug-and-Play Components for Evolutionary Software Development, In Proceedings of the Conference of Object-Oriented Programming, Systems and Languages (OOPSLA'98), 1998.

[12] T. A. Campbell, QoS Architectures, In Multimedia Communications Networks, M. Tatipamula and B. Khasnabish (Eds.), Artech House Publishers, Chapter 3, 1998.

[13] W. L. Hursch and C. V. Lopes, Separation of Concerns, Technical Report NU-CCS-95-03, Northeastern University, 1995.

[14] Y. Ichisugi, EPP: Extensible Type System Framework for a Java Pre-Processor, In Proceedings of SPA'99, 1999.

[15] Adam J. Lee, Jodie P. Boyer, Lars E. Olson, and Carl A. Gunter. Defeasible Security Policy Composition for Web Services. Technical Report, CS Dept, University of Illinois, 2005.

[16] Kevin D. Lux, Michael J. May, Nayan L. Bhattad, and Carl A. Gunter. WSEmail: Secure Internet Messaging Based on Web Services. Technical Report, University of Pennsylvania.

[17] TulaFale: A Security Tool for Web Services, http://blogs.msdn.com/bgroth/archive/2005 /01/11/351109.aspx

[18] ProVerif: A Tools for Analysis of cryptographic protocols, http://www.di.ens.fr/~blanch et/crypto-eng.html

[19] D. Damianou, N. Dulay, E. Lupu, and M. Sloman. The Ponder Policy Specification Language. In 2nd International Workshop on Policies for Distributed Systems and Networks, Bristol, UK, January 2001.

[20] T. Grandison and M. Sloman, Trust Management Tools for Internet Applications, Proc 1st Int.Conference on Trust Management, May 2003, Crete, Springer LNCS 2692, pp 91-107.

[21] L. Lymberopoulos, E. Lupu and M. Sloman. Using CIM to Realize Policy Validation within the Ponder Framework. DMTF Global Management Conference, San-Jose, California, June 2003. (winner Academic Alliance Paper Competition)

[22] A. Uszok, J. M. Bradshaw, R. Jeffers, N. Suri, P. Hayes, M. R. Breedy, L. Bunch, M. Johnson, S. Kulkarni and J. Lott. KAoS policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement. In Proceedings of Policy 2003, Como, Italy.

[23] M. Winslett, T. Yu, K. E. Seamons, A. Hess, J. Jacobson, R. Jarvis, B. Smith, , and L. Yu. Negotiating trust on the web. IEEE Internet Computing, 6(6), 2002.

[24] T. Yu, Automated Trust Establishment in Open Systems, PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 2003.

[25] T. Yu, M. Winslett, and K. E. Seamons, "Automated Trust Negotiation over the Internet," 6th World Multiconference on Systemics, Cybernetics and Informatics, Orlando, July 2002.

[26] M. Winslett, "An Introduction to Automated Trust Negotiation," Workshop on Credential-Based Access Control, Dortmund, October 2002.

[27] K. Seamons, M. Winslett, T. Yu, L. Yu, and R. Jarvis. Requirements for Policy Languages for Trust Negotiation. In 3rd International Workshop on Policies for Distributed Systems and Networks, Monterey, CA, June 2002.

[28] World Wide Web Consortium (W3C), http://www.w3.org/

[29] Organization for the Advancement of Structured Information Standards, http://www.oasis-open.org/home/index.php

[30] Web Services Enhancements (WSE),

http://msdn.microsoft.com/webservices/building /wse/default.aspx

[31] J. Fenton and M. Thomas. Identified internet mail. Work in Progress draft-fenton-identified-mail-01, IETF Internet Draft, October 2004. Expires April 2005.

[32] A. Serjantov and S. Lewis. Puzzles in P2P systems. 8th CaberNet Radicals Workshop, Corsica, October 2003. http://citeseer.ist.psu.edu/637647.html

[33] A. Back. Hashcash – A Denial of Service Countermeasure. 1997. http://www.cypherspace.org/hashcash/

[34] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In Ernest F. Brickell, editor, Proc. CRYPTO '92, pages 139-147. Springer-Verlag, 1992.http://research.microsoft.com/research/sv/PennyBlack/junk1.pdf

[35] T. Aura, P. Nikander, and J. Leiwo. DOS-resistant authentication with client puzzles. In Bruce Christianson, Bruno Crispo, and Mike Roe, editors, Proceedings of the 8th International Workshop on Security Protocols, to appear in the Lecture Notes in Computer Science series, Cambridge, UK, April 2000. Springer. http://citeseer.ist.psu.edu/aura00dosresistant.html

[36] D. J. Bernstein, *SYN Cookies*, http://cr.yp.to/syncookies.html

[37] P. Patel, A. Whitaker, D. Wetherall, J. Lepreau, and T. Stack. Upgrading transport protocols using mobile code. In Proc. SOSP, pages 1–14, Bolton Landing, NY, Oct. 2003.

APPENDIX A: WSEMAIL-PUZZLE DEMO SCREENSHOTS

This section contains screen shots of a WSEmail-Puzzle demo. The RMTA policies requires sender SMUA to solve a RTT puzzle and the RMUA requires sender SMUA to solve a Hash Cash puzzle. The Figures 20 and 21 shows the corresponding policies.

```
<WSEmailPolicy xmlns:xsd=http://www.w3.org/2001/XMLSchema
       type="static" node="WSEmailMTA">

<Ingresspolicy xmlns="http://securitylab.cs.uiuc.edu/WSEmail">
       <PolicyRules>
              <PolicyRule Id="#05" type="P" DllName="RTT.dll"
              RetrieveURI="http://seclab-london.cs.uiuc.edu/classes/RTT.dll"
              FriendlyName = "RTT No.1" Version = "1.0">
                     <Key>RTT</Key>
                     <Description>RTT Puzzle</Description>
              </PolicyRule>
       </PolicyRules>
</Ingresspolicy>

</WSEmailPolicy>
```

Fig 20: RMTA Policies

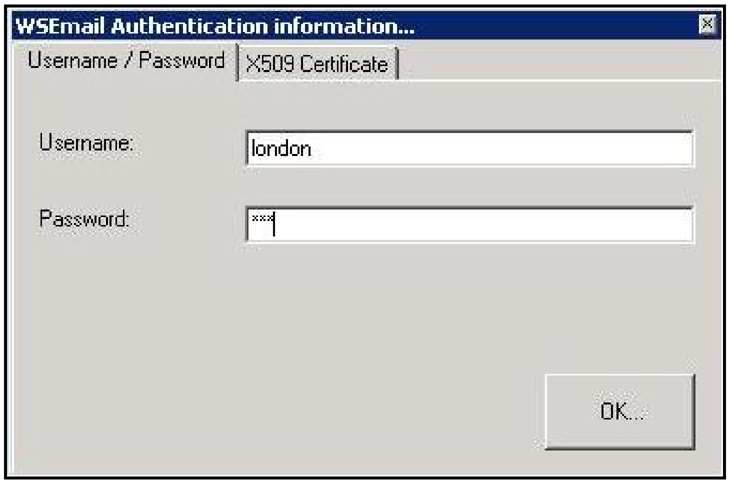
```
<WSEmailPolicy xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       type="static" node="WSEmailMUA">

<Ingresspolicy xmlns="http://securitylab.cs.uiuc.edu/WSEmail">
       <PolicyRules>
              <PolicyRule Id="#01" type="A" DllName="HashCash.dll"
              RetrieveURI="http://seclab-
              london.cs.uiuc.edu/classes/HashCash.dll"
              FriendlyName = "HashCash No.1" Version = "1.0">
                     <Key>HashCash</Key>
                     <Description>HashCash Puzzle</Description>
              </PolicyRule>
       </PolicyRules>
</Ingresspolicy>

</WSEmailPolicy>
```

Fig 21: RMUA Policies

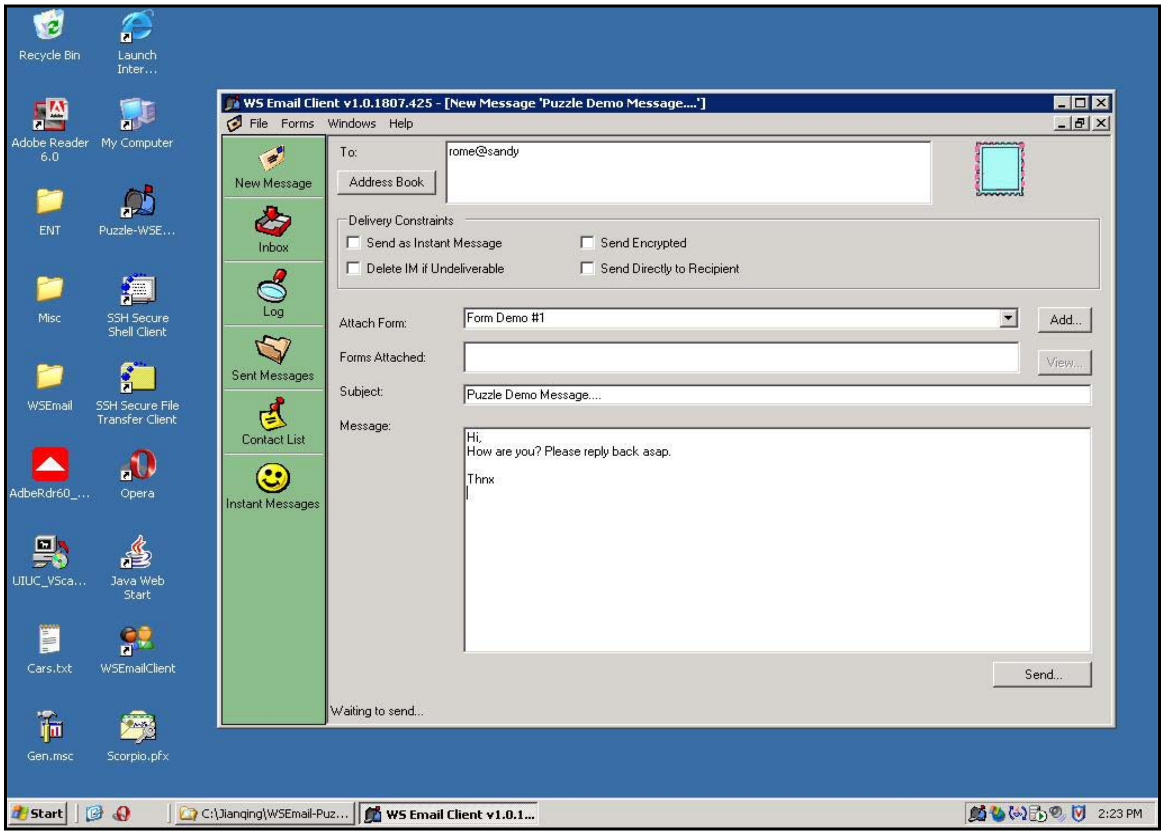
The following screenshots step by step shows a scenario of SMUA sending message to RMUA and solving puzzles required by RMTA and RMUA to successfully send the message.


WSEmail Authentication information...

Username: london

Password: ***

Fig 22: Login Window


WS Email Client v1.0.1807.425 - [New Message 'Puzzle Demo Message....']

To: rome@sandy

Delivery Constraints

- [ ] Send as Instant Message
- [ ] Send Encrypted
- [ ] Delete IM if Undeliverable
- [ ] Send Directly to Recipient

Attach Form: Form Demo #1

Forms Attached:

Subject: Puzzle Demo Message....

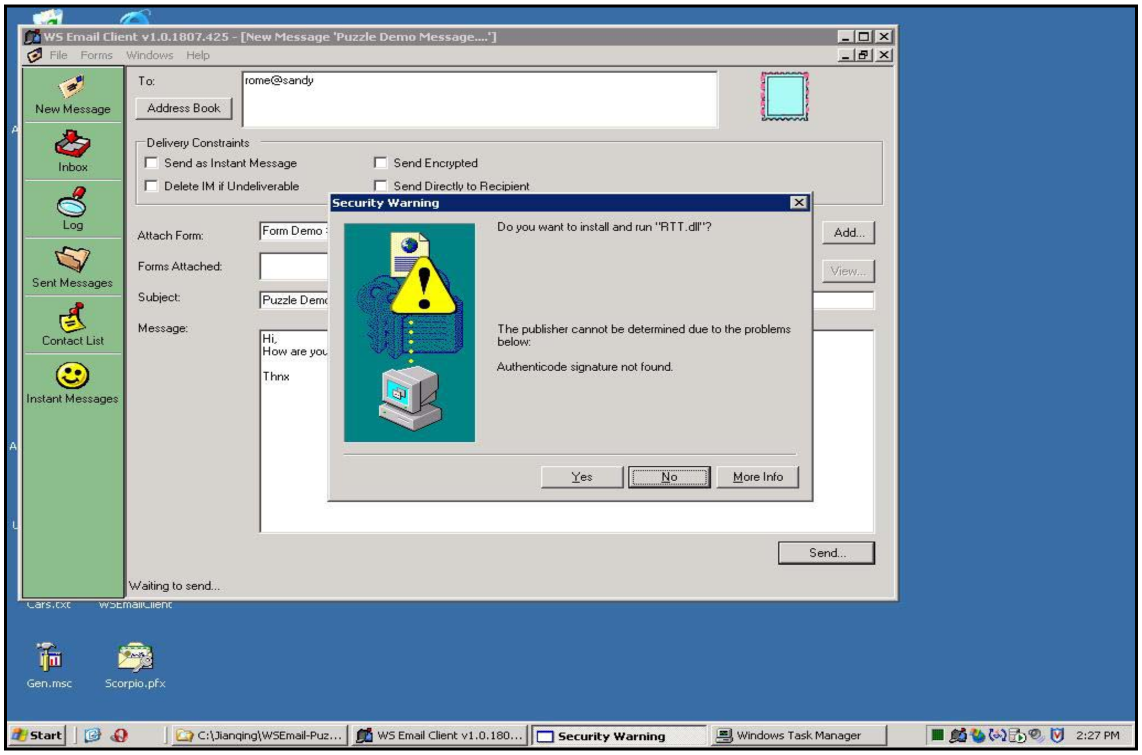
Message:

Hi,
How are you? Please reply back asap.

Thnx

Waiting to send...

Fig 23: Message Sending Window



To: rome@sandy

Delivery Constraints
Send as Instant Message
Delete IM if Undeliverable
Send Encrypted
Send Directly to Recipient

Attach Form: Form Demo
Forms Attached:
Subject: Puzzle Demo
Message:
Hi,
How are you
Thnx

Security Warning

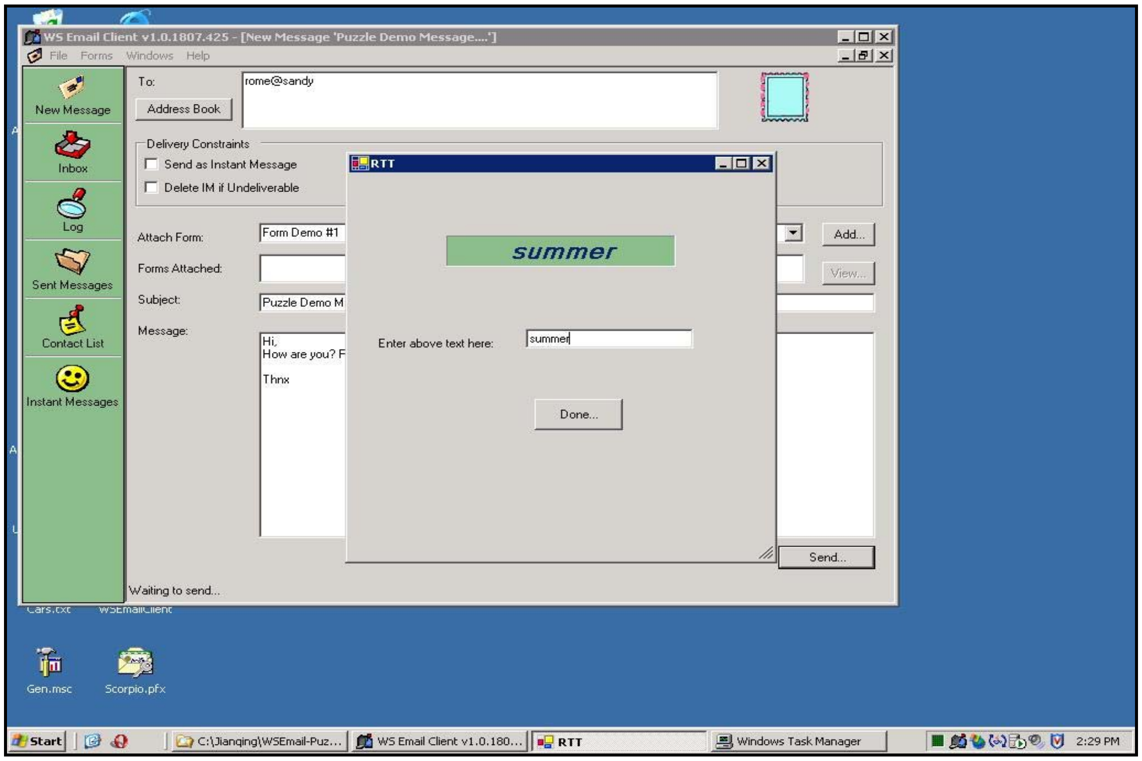
Do you want to install and run "RTT.dll"?

The publisher cannot be determined due to the problems below:

Authenticode signature not found.

Waiting to send...

Fig 24: RTT Puzzle Download



To: rome@sandy

Delivery Constraints
Send as Instant Message
Delete IM if Undeliverable

Attach Form: Form Demo #1
Forms Attached:
Subject: Puzzle Demo M
Message:
Hi,
How are you? F
Thnx

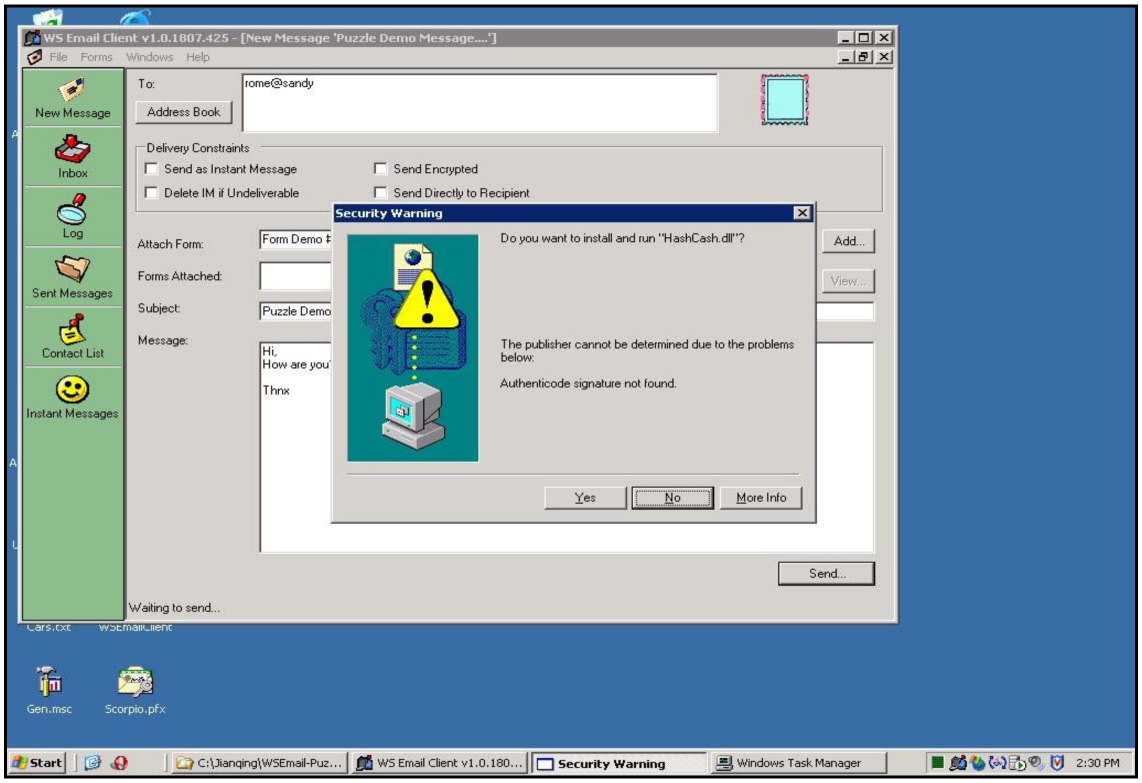
RTT

summer

Enter above text here: summer

Waiting to send...

Fig 25: Solving RTT Puzzle



WS Email Client v1.0.1807.425 - [New Message 'Puzzle Demo Message....']

To: rome@sandy

Delivery Constraints
- [ ] Send as Instant Message
- [ ] Send Encrypted
- [ ] Delete IM if Undeliverable
- [ ] Send Directly to Recipient

Attach Form: Form Demo #

Subject: Puzzle Demo

Message:
Hi,
How are you
Thnx

Security Warning

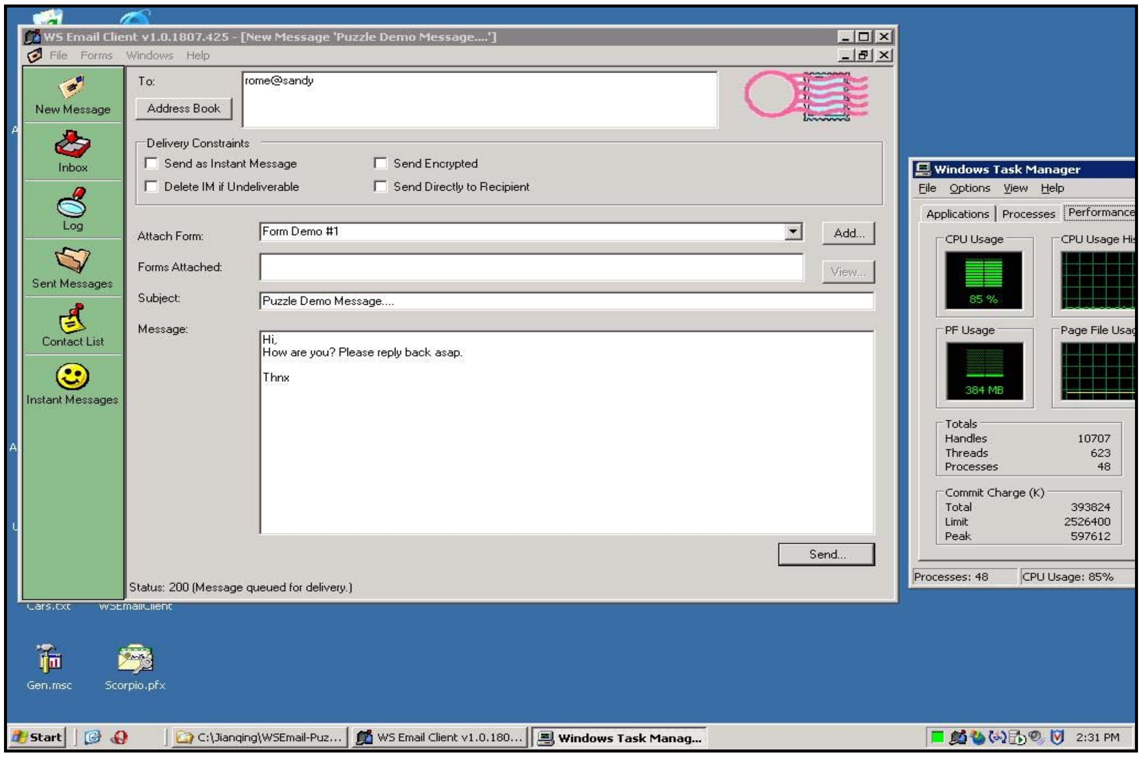
Do you want to install and run "HashCash.dll"?

The publisher cannot be determined due to the problems below:

Authenticode signature not found.

Waiting to send...

Fig 26: Hash Cash Puzzle Download



WS Email Client v1.0.1807.425 - [New Message 'Puzzle Demo Message....']

To: rome@sandy

Delivery Constraints
- [ ] Send as Instant Message
- [ ] Send Encrypted
- [ ] Delete IM if Undeliverable
- [ ] Send Directly to Recipient

Attach Form: Form Demo #1

Subject: Puzzle Demo Message....

Message:
Hi,
How are you? Please reply back asap.
Thnx

Status: 200 (Message queued for delivery.)

Windows Task Manager

CPU Usage: 85 %

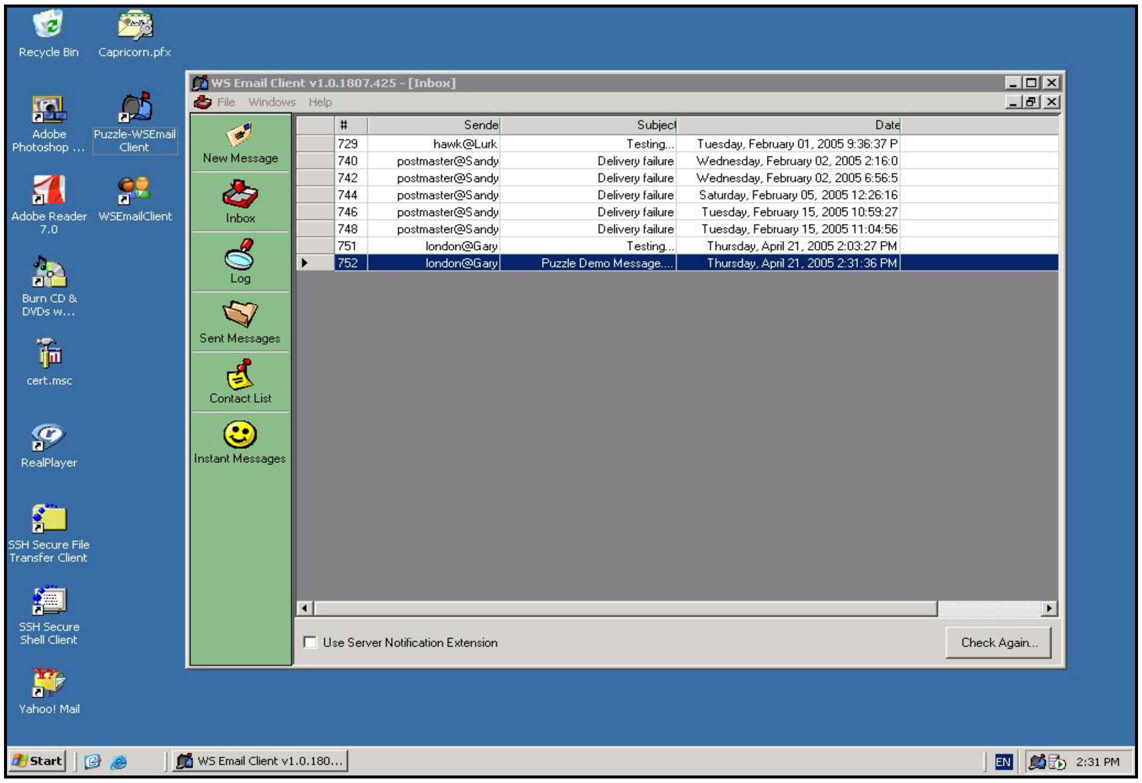
PF Usage: 384 MB

Totals	
Handles	10707
Threads	623
Processes	48

Commit Charge (K)	
Total	393824
Limit	2526400
Peak	597612

Processes: 48 CPU Usage: 85%

Fig 27: Solving Hash Cash Puzzle

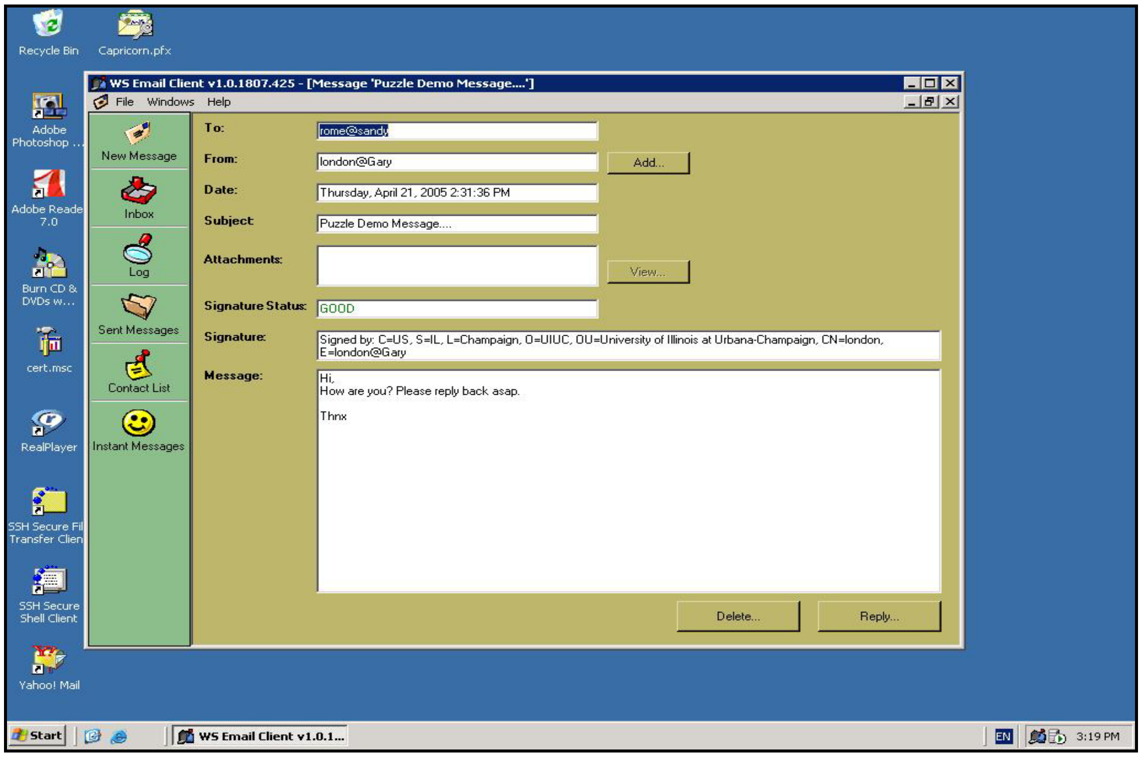


WS Email Client v1.0.1807.425 - [Inbox]

#	Sende	Subject	Date
729	hawk@Lurk	Testing...	Tuesday, February 01, 2005 9:36:37 P
740	postmaster@Sandy	Delivery failure	Wednesday, February 02, 2005 2:16:0
742	postmaster@Sandy	Delivery failure	Wednesday, February 02, 2005 6:56:5
744	postmaster@Sandy	Delivery failure	Saturday, February 05, 2005 12:26:16
746	postmaster@Sandy	Delivery failure	Tuesday, February 15, 2005 10:59:27
748	postmaster@Sandy	Delivery failure	Tuesday, February 15, 2005 11:04:56
751	london@Gary	Testing...	Thursday, April 21, 2005 2:03:27 PM
752	london@Gary	Puzzle Demo Message....	Thursday, April 21, 2005 2:31:36 PM

☐ Use Server Notification Extension

Fig 28: Message Received

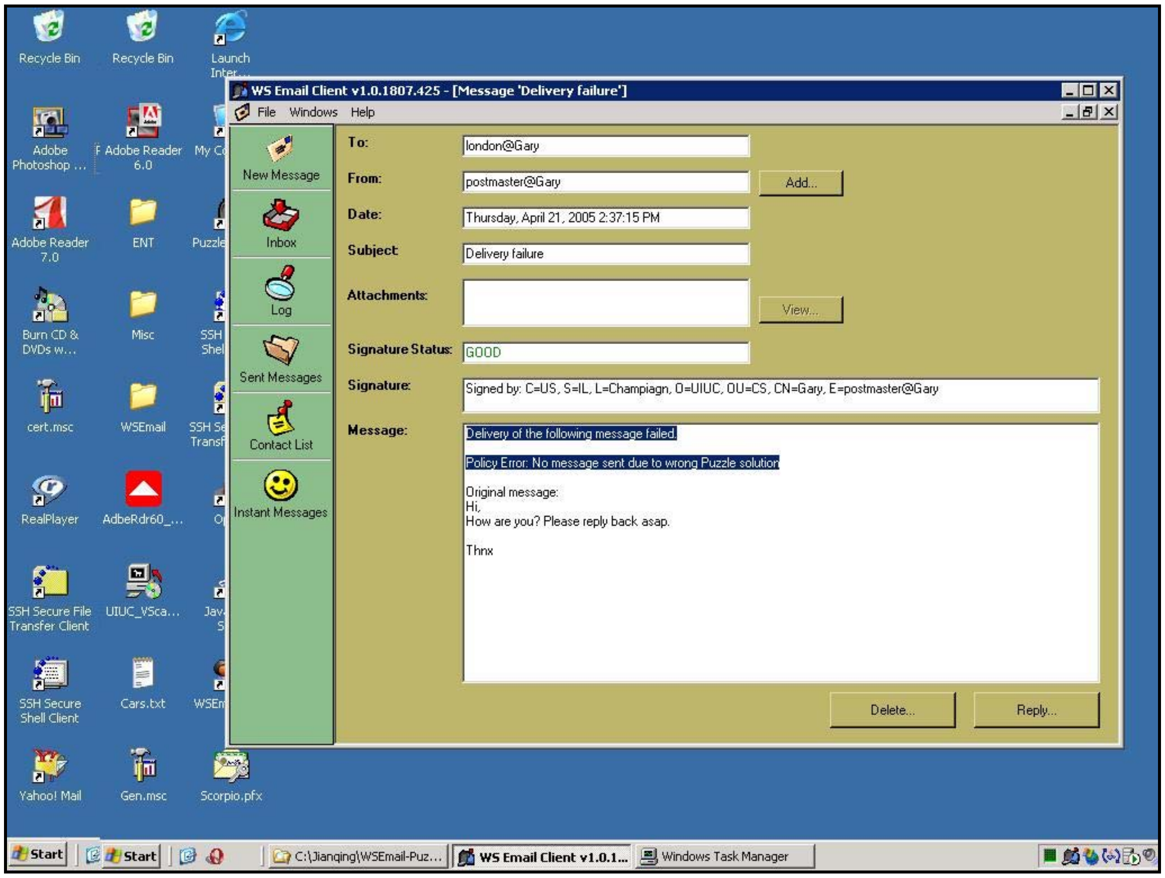


WS Email Client v1.0.1807.425 - [Message 'Puzzle Demo Message....']

To:	rome@sandy
From:	london@Gary
Date:	Thursday, April 21, 2005 2:31:36 PM
Subject:	Puzzle Demo Message....
Attachments:	
Signature Status:	GOOD
Signature:	Signed by: C=US, S=IL, L=Champaign, O=UIUC, OU=University of Illinois at Urbana-Champaign, CN=london, E=london@Gary
Message:	Hi, How are you? Please reply back asap. Thnx

Fig 29: Viewing Message

If sender is not able to correctly solve the puzzle then message is not delivered and "Delivery Failure" error is send back to sender.



To: london@Gary

From: postmaster@Gary

Date: Thursday, April 21, 2005 2:37:15 PM

Subject: Delivery failure

Attachments:

Signature Status: GOOD

Signature: Signed by: C=US, S=IL, L=Champiagn, O=UIUC, OU=CS, CN=Gary, E=postmaster@Gary

Message:

Delivery of the following message failed.

Policy Error: No message sent due to wrong Puzzle solution

Original message:
Hi,
How are you? Please reply back asap.

Thnx

Fig 30: Viewing Error Message

AUTHOR'S BIOGRAPHY

Raja N. Afandi graduated from the GIK Institute of Engineering Sciences and Technology, Pakistan in 2000 with Bachelors in Computer Systems Engineering. Before joining UIUC graduate programme, Afandi worked as a Software Analyst for ABN-AMRO Global IT Systems, Pakistan for several years. He completed a Master of Science in Computer Science from the University of Illinois in 2005.