Argus

Argus Design Specification
13 September 1989 at 22:43

Wendy E. Mackay
Brian R. Gardner
Toben H. Mintz
Richard A. Pito
Jeffrey B. Siegal

1. Problem Statement

Project Athena creates and processes a large volume of electronic messages using a variety of mechanisms, including *mh, rmail, discuss, BBoard, Zephyr* and *turnin/pickup* (for course assignments). Different groups need to monitor these messages for different reasons. Individuals who rely on personal electronic mail to prioritize their work may need help categorizing, organizing and finding messages. People who use *discuss* to discover technical information may need help finding relevant messages and retrieving them later. Small work groups may need help keeping track of bug reports, meeting announcements, project status, and who's working on what. Athena Operations receives many system status messages and may need help categorizing and responding to them, based on the current state of the system. Professors in large classes may need help keeping track of student assignments and allocating them to teaching assistants. Students may need help finding answers from old problem sets or in databases of answers created by T.A.'s.

Experienced programmers can use *grep* and *awk* to identify characteristics of messages and perform appropriate actions. Unfortunately, each program is different and non-technical Athena users do not have the expertise or patience to use these tools. Argus[1] provides a windowed interface to a rule system that gives users a simple way to categorize different kinds of messages, evaluate them and perform a range of actions based on those categorizations.

Users may create useful categories in a variety of ways. They may choose to look at the existing components of a message, such as the TO: field in a mail message. They may make decisions based on information calculated about the message, such as its length. Users can also base actions on previous processing of rules, such as detecting whether or not a message has already been moved. Senders may help to categorize messages, either by using pre-defined message types, such as BUG-REPORT or by adding extra fields, such as "Back-from-vacation: September 5, 1989". Recipients of messages can also categorize messages, either by adding extra fields, such as "Delete-in-1-week: September 5, 1989" or by using an Argus rule action, such as "set property: important".

Once messages have been categorized, Argus rules can perform different actions. Rules can operate on the messages themselves, such as sort, modify, delete, redistribute or convert to a different type of message. They can also perform other tasks, such as call Unix programs, generate new messages or run other rules. Rules are stored in groups, called rulesets, and are run sequentially over a stream of messages. Rulesets may be run by users, automatically triggered by certain events, called from programs or invoked as a rule action.

Argus is not a message system, but rather a method for handling electronic messages. It gives users a simple, standardized method for categorizing and responding to electronic messages and can be used for a variety of educational and system management applications. Argus is designed to support extensions to the formats of messages that can be handled, how those messages are categorized (rule predicates), and the actions that can be performed (actions).

---

[1] Argus is, unfortunately, the name of another well-known system developed at MIT. We are looking for other names and welcome any suggestions.

2. Proposed Athena Applications

Project Athena has a variety of users, each with different sets of needs. Although the original purpose of Argus was to help Athena users manage their electronic mail, the scope has been extended to handle categorizing and processing of a variety of different kinds of electronic messages. Some of the following application suggestions can be implemented immediately, others require a few enhancements. Chapter 9, Proposed Extensions, describes applications that will require more significant effort.

Athena's educational mission gives students the opportunity to try simulations and work through problem sets, which are submitted electronically to professors and teaching assistants. They use a variety of message systems, including electronic mail, *discuss* and *turnin/pickup* to exchange information with each other and to hand in and receive assignments. Most of these messages have internal structures that can be accessed by Argus.

Some professors use electronic mail or *discuss* to explore issues and answer questions. Some times previous problem sets, quizzes, or answers to common questions are made available to students. Argus could provide students the ability to gain access to this information at any time, without requiring the T.A. or professor. Many of the features of *Common Ground* can be accommodated by Argus, in conjunction with *discuss*.

Argus can be used to help T.A.'s and professors in large classes manage the exchange and grading of problem sets. One proposed extension would be to use the Argus rule editor as a front end to the distributed *turnin-pickup* programs, to provide a more powerful tool for identifying and tracking the administrative side of courses.

A special feature of Argus is its ability to collect data about its use. Students in the Managerial Psychology Laboratory (15.301) could use it to monitor their communication with each other, as part of their study of communication systems. Students can also use it to find out about their own use of mail and other message systems and compare it to their perceptions of use. The data is stored as messages, which can then be interpreted with Argus rules.

Athena staff and course development groups may also benefit from Argus. Groups that exchange information electronically, such as documentation or systems development, may find that using a few pre-defined message types or extra fields helps them manage the flow of information. For example, A software development group could agree on a certain format for bug reports. Users could send these bug reports (or the person who reads them could mark them consistently). The programmers could then respond to messages in order of priority and bug reports could be stored according to their type and importance. Groups can also informally agree to use their own group-specific message header fields to facilitate communication with each other. For example, if people want to arrange a time for a meeting, everyone can propose meeting times in a "next-meeting" field, which can then be extracted and processed automatically.

The visitor center could respond to electronic requests for demos with a standard "visit-request" template. With the messages in a standard format, it would be easy to write different rules to find different messages, create schedules automatically, or clean up old files. Operations could use messages in a similar way, by responding to equipment requests with standard forms. These could then be tracked on-line. Users might even be able to send messages and get automatic replies about the status of their requests. Another application would be to monitor system status messages. If a certain error condition occurs, a rule could run that notifies someone or causes another service to take over. The ease of creating rules would make it easy for system administrators to accommodate on-going changes in the system. Another potential benefit is the ability to analyze which rules have fired, providing a summary of the kinds of messages that have occurred and possibly providing diagnostic information.

Athena will be gaining access to more on-line services, such as the library. Argus can help students find abstracts of articles of interest and create useful on-line bibliographies.

Finally, Argus can be used for its original purpose: to help manage the flow of electronic mail. Staff members and students in classes in which there are large quantities of electronic mail can use Argus to find the most important messages, redistribute them to key people, save messages for easy subsequent retrieval, and delete messages that are unimportant or out of date. Mh users can use variations of the *anno* (annotate) command, as they read their mail, to mark messages for subsequent processing. For example, a user may mark a vacation announcement with "delete-in-1-week" and a request for information with "remind-me-in-2-weeks". Rules can then be used to automatically delete or send zephyr messages as reminders.

3. Architecture

Overview

Argus handles two basic forms of information: streams of messages and sets of rules. **Messages** may come from a variety of sources. Argus currently handles two storage schemes for electronic mail: *mh* and *babyl*, which allows users to use it with *mh, Xmh, Rmail,* and *babyl*. Extensions to other electronic messages, such as *zephyr* and *discuss* can be accommodated by creating a separate translation program for each. (The translation program creates a standard, internal message format that can be understood by all Argus programs.) Messages are assumed to be ASCII text, usually with some internal structure. A group of messages is called a folder, and may be an open-ended stream or a closed file or group of files.

The translation program must be specified when applying a set of rules to a group of messages. (Argus does not detect the format of the source of the messages.) The translation program extracts message header fields and information specific to the source of message and creates a standard internal data structure that is used by all Argus programs. Chapter 5 provides a more detailed description of messages. Messages can contain five kinds of components:

1. Text fields from the message (e.g. the TO: field in a message).
2. Properties defined by users (e.g. "important").
3. Properties calculated about the message (e.g. number of lines).
4. State of rule processing (e.g. whether a rule has fired on a message).
5. Properties specific to the source of the message (e.g. message has been replied to).

Users create **rules** with a rule editor. The rule language uses post-fix notation and is designed to be readable by users, easy to read into the Rule Editor, and fast to execute. Groups of rules are stored as ASCII text in user files called rulesets. The user can specify how to interpret each component of a message and what operators to apply. (For example, a rule may specify that a DATE: field is to be interpreted as a date and compared to the specified value with the operator *before*.) The rule system evaluates the message components and, if true, performs the actions specified.

Rules have a predicate (IF) and one or more actions (THEN). Each predicate line in a rule specifies a message component and its type, the operator to apply, the way the message component is to be

example, a rule might include the following line:

```
IF  DATE:  before today
```

where DATE: is specified as a field from the message and *today* is to be interpreted as a date.

A message might have the following field in its header:

```
Date:  Thu, 24 Aug 89 14:27 EDT 1989
```

If the rule is run on 26 August, 1989, the rule will:

1. Look for a message field DATE:. (Here, it finds it.)

2. Look for a text string after the field. (Here, Thu, 24 Aug 89 14:27 EDT 1989.)
3. Interpret the text string as: Thu Aug 24 14:27:00 1989 and convert into seconds.
4. Interpret *today* as: Sat Aug 26 20:28:57 1989 and convert into seconds.
5. Evaluate whether or not Thu Aug 24 14:27:00 1989 is before Sat Aug 26 20:28:57 1989.
6. Set the predicate line to True.

Each rule in a ruleset is applied in turn to the same message. The next message in the folder is then retrieved and the ruleset begins again with the first rule. Chapter 4 provides a more detailed description of the rule system and chapter 6 provides a more detailed description of the rule evaluation process. Figure 3-1 shows a simple overview of how Argus evaluates a a stream of messages against a set of rules.

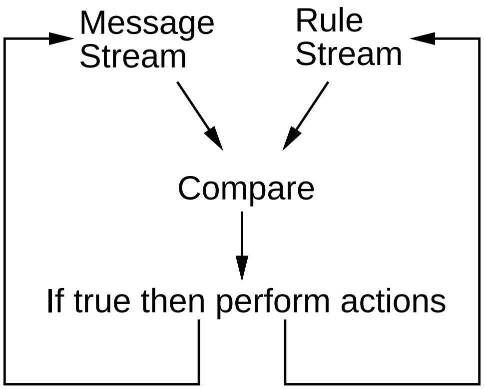

Figure 3-1: How Argus handles streams of messages and rules

Argus Structure

Argus consists of three types of code: modules, data structures, and resources. Modules include the various Argus programs, such as runrules and the rule editor, parsers, data collection routines, and translation modules. Modules operate on and exchange several kinds of data structures, including: rules, rulesets (streams of rules), messages, folders (streams of messages) and message types. Resources help to manage data structures in a consistent way, and include: registries, files, message syntax, and user interface guidelines. Figure 3-2 shows a summary of the Argus modules, data structures, and resources.

Users perform three basic activities with Argus: they create or modify rules, they run rules on specified groups of messages, and they change user-defined parameters of the system. These activities are supported by a set of programs, which can each be called from the Unix command line. (Eventually, many of them will also be available from the Argus control panel.) See Appendix VII for the usage listing for each program. The current set of Argus applications include:

rulebrowser (Section 4.2.3) Users can examine a set of rules, print them, rearrange them, and

Data Structures	Modules	Resources
Message Message Type Rule Ruleset Attribute (field) Actions Data	Message Type Editor Alcomp Rule Editor Rule Browser Runrules Alinc (mh) Data Controller Annotator Rule Pretty Printer	Message Type Registry Ruleset Registry Verb Registry Folders Profile Xresources Mailer interface

Figure 3-2: Argus Structure: modules, data structures and resources

choose which ones to edit, move, copy or delete. The ruleset specification is required. To browse the ruleset named "siftrules" a user would type:

`rulebrowser siftrules`

ruleeditor (Section 4.2.4) Users can create a rule and specify where to store it in a ruleset. To create a rule, users type values into message fields. Menus provide alternative message components and operators for the IF part of the rule and alternative actions for the THEN part of the rule. The ruleset specification is required; the rule number and message type are optional. To edit the first rule in the ruleset named "siftrules" a user would type:

`ruleeditor -ruleset siftrules -rule 1`

runrules (Section 6.1) Users can run a specified ruleset over a specified stream of messages (or folder). The ruleset and message stream are required. Message source, data collection options, whether or not to save user property and state information, and method of categorizing rule processing are all optional. To run the ruleset named "siftrules" over an mh inbox, a user would type:

`runrules siftrules inbox`

alinc (Section 6.1) Mh users can run a specified ruleset while *inc*'ing new mail. The ruleset specification is required; the mail folder defaults to *+inbox* if none is specified. To *inc* new mail and run the ruleset named "incrules" over them, a user would type:

`alinc incrules`

alcomp (Section 5.3) Mh users can compose messages with message type fields. The message type is optional. To compose a message of the type "phone", a user would type:

`alcomp -msgtype phone`

argusi (Section 7.3) Users can specify what types of data to collect about how they run rules, including: rule life, rule activity, and message traffic. During the user study, the default is for data collection to be on. No options are available. To run argusi, a user would type:

`argusi`

mteditor (Section 5.2) Users can create or review message types, which are collections of message fields. If no message type is specified, the message type *MSG* is used. To run the message type editor, a user would type:

`mteditor -msgtype phone`

4. Rule System

Rule Language: Formal Description

The rule language is based on a postfix format. Keywords are identified with a dollar sign: `$if`. User-supplied text, fields or properties from a message and component types appear in quotation marks: `"To:"`. Operators are enclosed in quotes and preceded with a percent: `"%equals"`. Booleans that group predicate lines are preceded with a percent: `%or`. Actions are preceded by a percent: `%move-to`.

Quotes are used in the predicate line so that spaces can be included for any of the component values. For example, a field might contain the value "very important". The rule parser relies on the error checking done in the rule editor and assumes a valid ruleset. It does not do separate error checking of its own. The following is an annotated, formal description of the rule language. (See Figure 4-1 for a list of built-in components, operators, and types.)

Ruleset header:

$format_version <integer>	***Currently version 1***
$desc <optional-string>	***Ruleset descriptor***
$created <time (Unix format)>	***Ruleset creation date***
$updated <time (Unix format)>	***Ruleset updated date***

Repeat for each rule:

$created <time (Unix format)>	***Rule creation date***
$updated <time (Unix format)>	***Rule updated date***
$name <optional-string>	***Rule name***

$if ***List of predicate lines. Groups may be joined with a boolean and indented together.***

"<Component>:" "%<Operator>" "<Value>" "<Type>" "%<integer>"

Component:	***Field, state or property of message.***	***(TO:)***
Operator	***Compare component to value in msg.***	***(contains)***
Value	***Value of component. Ignore if empty.***	***(staff)***
Type	***How to interpret a message component.***	***(string)***
Integer	***The kind of component (5 possible).***	***(msg field)***

%<boolean> <integer>	***Specifies boolean and number of predicate lines in the group.***	***(OR)***
%and <integer>	***Normally <integer> = 1***	

$then ***list of action lines:***

[<Arg$_1$> [<Arg$_2$> [..<Arg$_n$>]]] %<Verb> <Num-args>

Arg$_i$	***Argument to the action***	***(staff)***
Verb	***Verb***	***(move-to)***
Num-args	***Number of arguments in action line***	***(1)***

$end

Example of a rule in the rule language

The following is an example of a rule that moves messages addressed to staff or copied to staff to a separate folder called "staff".

```
$format_version 1
$desc <no description>
$created 619398053
$updated 619398242

$created 619398242
$updated 619398242
$name Siftstaff
$if
   "To:" "%contains" "staff@athena" "Any-Case-Text" %0
   "Cc:" "%contains" "staff@athena" "Any-Case-Text" %0
  %or 2
 %and 1
$then
 "staff" %move-to 1
$end
```

The user would see this rule as follows (using the rule pretty printer):

Ruleset: Sifter

Rule 1: Siftstaff

If

	To:	staff@athena
OR		
	cc:	staff@athena

Then

move-to	staff

Rule Components

Predicates

The predicate, or "IF" part of a rule, evaluates the components of a message and interprets them based on the type specified by the user. Argus can handle the following five kinds of message components: (Figure 4-1 shows examples, the kinds of options the user can specify, and the data types associated with each message component.)

Message Field Field that appears in the message. May be provided automatically in the message header, typed in by the sender, added automatically by the receiving post office or typed in later by the recipient of the message. Examples include: TO: or DATE: fields of a mail message, or a serial number in a bug report.

User Property Created by rules as temporary or permanent variables which can be used or modified by subsequent rules. This is the only component that can be explicitly altered by the actions of a rule. Examples include: level of importance, classification codes, or a date by which the message should re-surface as a reminder.

Processing State Updated as a result of running rules. Examples include: the number of rules that have fired on this message, the number of times it has been moved, and the last folder from which it was moved.

Source Property Information specific to the source of the message (e.g. *mh*). Enables users to take advantage of properties unique to a particular message format. Examples include: mail systems that keep track of whether or not a message has been read, forwarded or replied to.

Message property Information calculated about the message or about fields within the message. Examples include: number of lines in message, number of unprintable characters in body, number of fields in a message and whether or not a field exists.

Message components are interpreted as one of the following types: strings, integers, booleans and dates. If new types are added, this list will be expanded. Text strings can be case-sensitive or not. The user can also specify that a comma is to be interpreted as an "or", which is particularly useful with lists of user names, such as `mackay, rap, thm`. The date parser is based on a Unix program called *getdate* and can handle a wide range of date specifications. The date parser translates all units of time into seconds. This causes problems for statements such as DATE: today, which should fire for all of the seconds in the day, not just the first one. To solve this problem, users must specify a unit of time as part of the type. Figure 4-2 lists the operators available for each data type.

Actions

Each rule can have one or more actions. Rules can also run other rulesets, which permits complex rule actions and the equivalent of IF-THEN-ELSE constructs. A ***stop*** action is provided to prevent further processing of a message if some condition is reached. Rules can also invoke shell scripts or Unix commands, such as *zwrite*. The message can be piped as standard input to the command. Systembody uses only the body of the message, which is useful for printing mailed postscript files. Variables from within the fields in the message can be passed to the shell command. Actions may take 0, 1 or 2 arguments. Figure 4-3 shows the current list of actions. A plus (+) after the number of arguments

Message Fields	User Properties	Message Properties	Processing State	Source Properties

Examples:

To: staff	Priority: 1	Total lines in msg: 30	Rules fired: 2	Replied-to: True

Alternatives:

Specified by user	Specified by user	Total fields in msg Total lines in msg Total lines in body Total characters in msg Total characters in body Total unprintables in msg Field count	Moved from Total times moved Copied from Total times copied Deleted Total rules fired Message source Message type	Specified by user

Operators:

Text strings** Date Time Integer Boolean Exists	Text strings** Date Time Integer Boolean Exists	Integer Boolean Exists	Integer Boolean Exists	Text strings** Date Time Integer Boolean Exists

**Text strings can specify four conditions:
Case Sensitive, comma = comma
Case Insensitve, comma = comma
Case Sensitive, comma = OR
Case Insensitive, comma = OR

Potential sources:
mh (xmh)
Rmail (babyl)
zephyr
discuss
turnin/pickup

Figure 4-1: Components of a Message: Fields, States and Properties

Text String **	Date	Time	Integer	Boolean
Contains word Word = Equals => Greater than =< Less than != Not equals Exists	= Equals != Not Equals < Before > After Exists	= Equals != Not Equals < Before > After Exists	= Equals != Not Equals < Less than > Greater than =< Less than or = => Greater than or = Exists	= Equals != Not Equals Exists

**Text strings can specify four conditions:
Case Sensitive, comma = comma
Case Insensitive, comma = comma
Case Sensitive, comma = OR
Case Insensitive, comma = OR

Figure 4-2: Types of components and their associated operators

indicates that one or more additional optional arguments are available.

Action	Arg's	Description	Example
Move-to	1+	Move message to specified folder. Set message state to *moved.* Optional: specify mailer type.	Move-to inbox Move-to inbox rmail
Copy-to	1+	Copy message to specified folder. Set message state to *copied.* Optional: specify mailer type.	Copy-to inbox Copy-to inbox rmail
Delete	0	Delete message. Set state to *deleted.*	Delete
Remail	1	Forward message to address list.	Remail mackay
Send-me	0	Send message (external server) to me.	Send-me
Runrules	2+	Run specified ruleset on specified folder. Optional: specify mailer type.	Runrules inrules inbox Runrules inrules inbox mh
Stop	0	Stop processing rules in current ruleset.	Stop
Set property	1+	Set user-defined property. Optional: give property a numberical value.	Set important Set important 10
Clear property	1	Clear (remove) specified user property.	Clear important
Add-to	2	Add integer to specified user property.	Add important 3
Subtract-from	2	Subtract integer to specified user property.	Subtract important 3
Permanent	0	Save all user properties on disk.	Permanent
Temporary	0	Delete user properties after running rules.	Temporary
System	1	Execute specified Unix command. Pipe message as standard input.	System zwrite
Systembody	1	Execute specified Unix command. Pipe body of message as standard input.	System lpr
Shell	1	Execute specified shell command. Optional parameters: /n new line /t tab %m message field %u user property %w whole message %b message body %h message header	Shell zwrite "Long Msg: /n %h"

Figure 4-3: Rule Actions: Definitions and Examples

Rule Editor

From a user's perspective, it is important that that rules be easy to understand; it should be quick and easy to create the most commonly used rules. The Rule Editor provides a template for specifying the conditions under which a rule will fire. The rule editor appears with a set of message fields, including TO: and CC:, each followed by an empty text field. In order to create a rule that fires if a message is addressed to "Staff@athena" and moves it to a folder called "staff", the user simply fills in the TO: field with "Staff@athena", chooses the "move-to" action from the "action" menu, and fills in "staff" in the empty text area.

A rule can refer to more than one field, state or property. If more than one component is specified, the default is to combine the fields with AND. However, the user might want to evaluate these with a different boolean operator. For example, a user might want to create a rule that fires if the message is addressed to *or* copied to "Staff@athena". The user can choose "add a box", select OR, and draw a box around the TO: and CC: fields. Figure 4-4 shows an example of this rule. The rule editor allows users to place a box around any fields and specify any combination of AND, OR or NOT.

The user can add extra fields with the "add a field" button. For example, the user might want to see if there's a "back-from-vacation" field in a message. In this case, the user would ask to evaluate the field as a date, not a text string. The user can also modify fields to specify user properties, processing states, source properties, and message properties (described in section 4.1.1). Menus of alternatives are available for processing states and message properties.

The user can evaluate whether or not a message is of a particular type, such as "bug-report". Rather than adding each field by hand, the user can choose the "message type" button and select from a menu of possible message types. When the rule is done, the user presses the "save" button, which inserts the rule into the specified ruleset.

Rule Browser

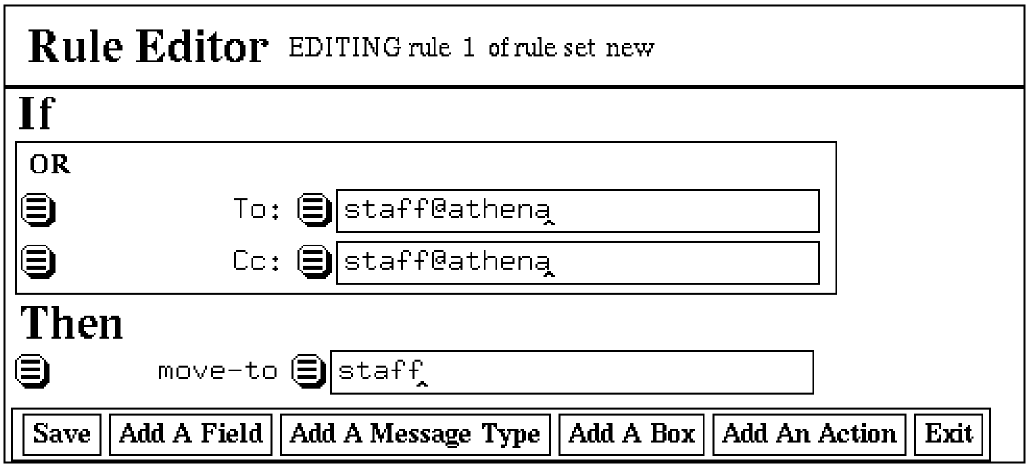

Figure 4-4: Rule Editor: Sample rule to move messages sent to or copied to staff to a folder called **staff**.

The Rule Browser allows users to display the rules in a ruleset on the screen and scroll through each individual rule. Users can print all the rules or select a particular rule to edit (by calling up the Rule Editor), copy, or delete. Users can add new rules (again with the Rule Editor) and specify where to place them in the ruleset. The Motif version will allow users to rearrange rules and call more than one Rule Editor at a time. Figure 4-5 shows a ruleset with the rule created with the Rule Editor in Figure 4-4.

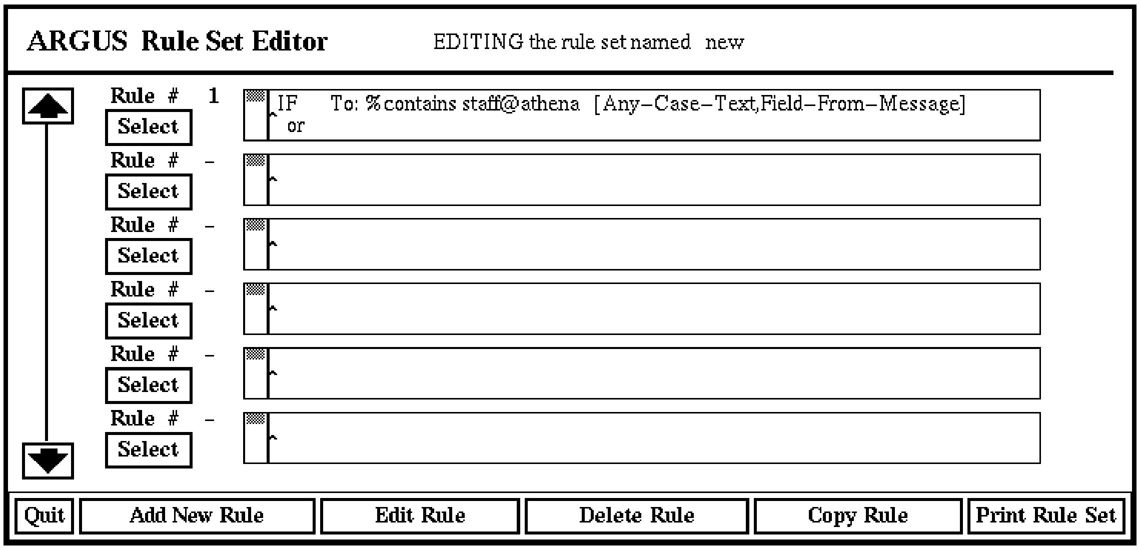

Figure 4-5: Rule Browser: Display of ruleset New

5. Messages

Messages are ASCII text files (or parts of files) available on the network. Figure 5-1 shows how Argus stores the components of a message in an internal data structure.

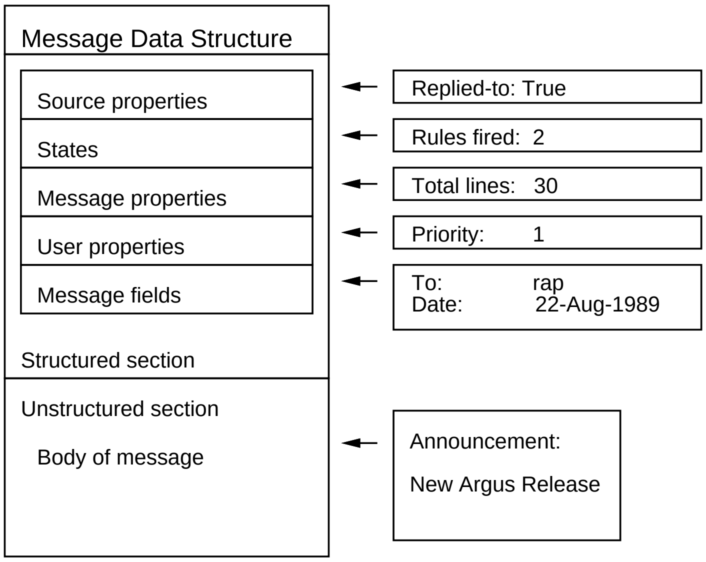

Figure 5-1: Definition of a message

Argus was designed to be extended to handle different kinds of electronic messages. The following sources of messages may prove useful to Project Athena:

mh Contains header fields of the form: <field>: and a value. Some of the fields are provided by the sender of the messages, others are added automatically by the mail system. Each message is stored in a separate file. Mh and Xmh provide access to functions such as move, delete, copy, and create new directories, which Argus uses directly.

rmail All messages are stored in a single file, in *babyl* format. Has the same header fields as *mh* mail messages. Keeps additional information about how the message has been processed, such as "replied-to: true".

zephyr Messages are broadcast and not stored, so Argus must interpret them as an open-ended message stream. Otherwise, format is similar to electronic mail.

discuss Messages are stored in a data base. Message headers are similar to mail messages, but also include indices that point to other messages.

turnin/pickup The message consists of a hashed file of headers, created by the *turnin/pickup* program and a "body", which is any file stored on one of several servers.

Translation of Messages

Messages are translated from one of the above kinds of messages into an internally-consistent message data structure (Figure 5-1). Argus can receive messages from virtually any source, as long as an appropriate translation module has been created. Rules can specify the output format of a message, which provides an easy mechanism for translating messages from one format to another.

The document entitled "A Programmers Guide to Creating Argus Translation Modules" provides a working example of a translation module (for mh) and explains how to create a new one. Experience in C and Unix is required. When creating a new translation module, the programmer must specify:

1. How to read the stream of messages (e.g. refer to a mail folder for mh, but open a socket to listen for messages with Zephyr).
2. How to identify the components of standard messages (e.g. message fields).
3. How to preserve information that is unique to that source of message. (For example, babyl mail files retain information about whether a message has been replied to or not. Argus can operate on such information through message-source properties.)

Message Types

The Message Type Editor allows users to create message types, which are pre-specified collections of fields, and store them in a central location. (Currently, the argus afs locker.) Examples of message types useful to Athena might include: "Vacation", "Phone", "Meeting announcement", "Bug Report", "Equipment Request" and "Visitor Center Request". Courses might specify "Question", "Problem Set", "Schedule Announcement" and "Answer". Figure 5-2 shows an example of a "Bug-fix" message type. Appendix IV shows the current set of message types, which is located in the argus *lib* directory.

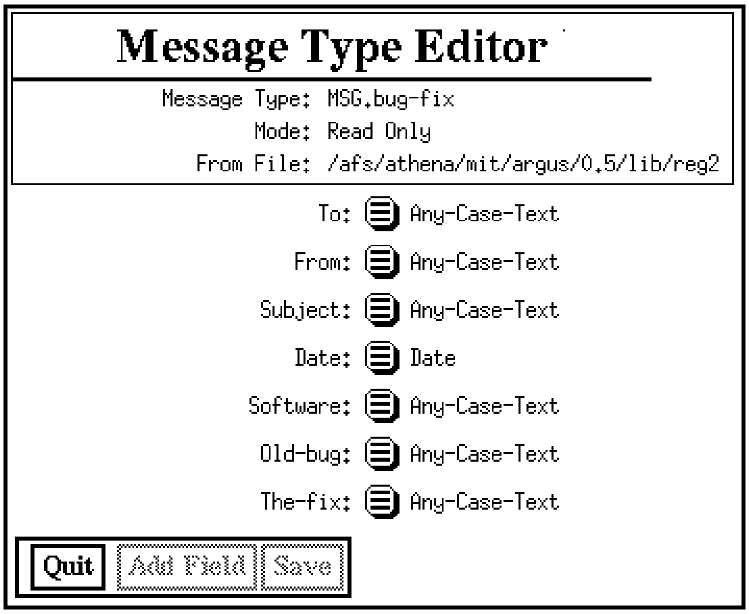

Figure 5-2: Message Type Editor: Bug-Fix message type

Alcomp

Alcomp was created to let *mh* users create messages with pre-specified message type fields. Of course, users can always add their own fields as well in any text editor or with the *comp* command. Alcomp takes the same arguments as the *comp* command, as well as the arguments of the Argus ***runrules*** command, and a message type. It then creates an emacs window with the fields specified in the message type. The example below shows how a user would create the bug-fix message type, which is illustrated in Figure 5-3.

```
alcomp -MTmessagetype MSG.Bug-fix
```

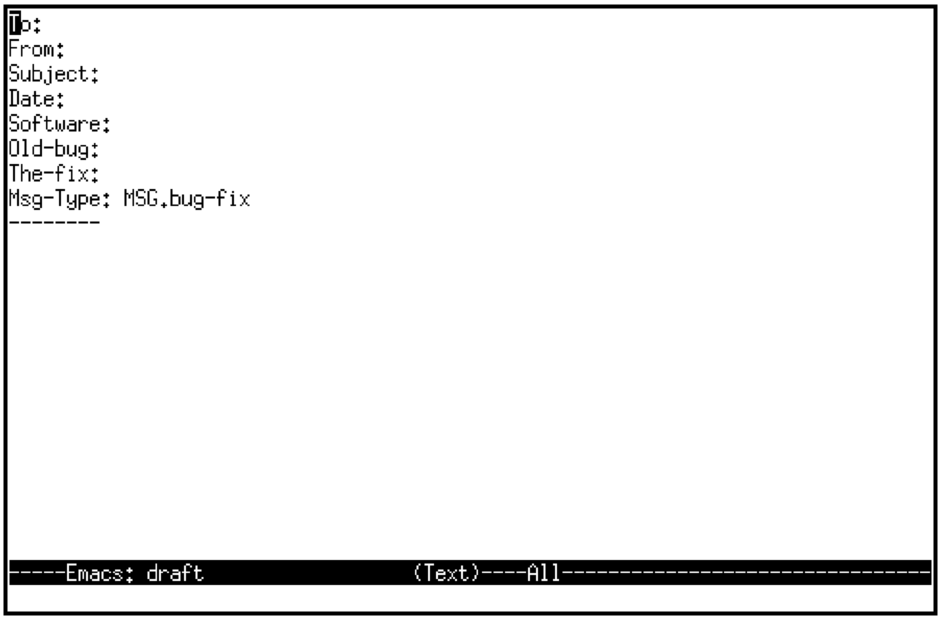

Figure 5-3: Alcomp: Editing a bug-fix message in Emacs

Annotating Messages

Sometimes the receiver of the message likes to be able to categorize it for later processing. For example, identifying messages that should be automatically deleted in a few weeks. Users can use rules to add user-defined properties to messages or they can use the Xmark command (for *mh* only) to annotate messages. This command appends a text string, followed by a colon and the current date and time to the top of the message. For example, if someone sends a vacation announcement, with a telephone number for where to find the person for the next week, the user might want to place a "1-week-delete: <today’s date>" field in the message. Later, a rule could fire that checks to see if a week has passed, and if so, deletes the message. Commonly used marks can be aliased to short commands, as shown in the example below.

```
d1
```

(where d1 is aliased to: alias d1 ’anno -component 1-week-delete:’)

6. Evaluating Rules

Figure 6-1 illustrates the process of evaluating streams of messages and rulesets. The order of processing can be summarized as:

$$\sum_{i=1}^{n}\sum_{j=1}^{m} rule_j(msg_i)$$

where $rule_j$ is the number of the rule and msg_i is the number of the message.

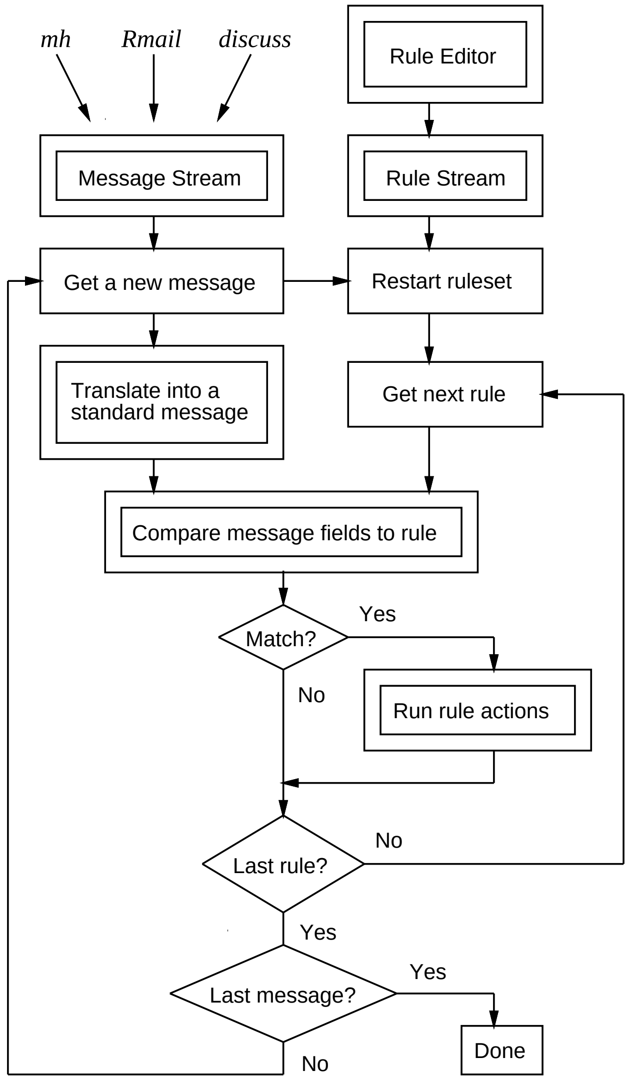

Figure 6-1: Rule Firing Process
Note: Double boxes indicate a special section in this document.

The process can be summarized as follows:

1. A user uses the rule editor to create one or more rules and stores them in a ruleset.
2. The user applies a ruleset to a folder (or stream of messages), with the *runrules* command. (Section 6.1.)

3. The rule system locates the ruleset, usually in the user’s *~/argus/rulesets* directory.
4. The translation module finds the messages (a mail file, a stream of messages, etc.) and translates the first message into a standard internal message format.
5. The rule system retrieves the first message from the folder and gets the first rule from the ruleset.
6. The rule system then extracts components from the message and evaluates it according to the rule.
7. If the rule fires, the action or actions specified in the rule are executed. No action is performed if the rule does not fire. (In other words, it is not considered an error.)
8. The rule system tests the next rule in the ruleset against the same message, unless a STOP action was specified in the previous rule. The rules are processed sequentially until the end of the ruleset.
9. After all the rules in the ruleset have been tested, the next message is retrieved from the folder and processed in the same way. This continues until all messages have been processed.

Destructive operations such as "delete" are deferred until all the rules have been tested. However, setting of user properties and states occurs in sequence. This gives all rules access to the original message, bu allows urles to react to changes in user properties or processing state.

Runrules

Runrules is a program that can be run at any time from the Unix command line. It may may also be called as an action within a rule, from another Unix program, or from a shell script. ***Runrules*** can also be used as a filter, pipe, or forked process. The user must specify which stream of messages to use (a *folder*). If no folder is specified, Argus will read messages from standard input (stdin). If no format is specified, Argus will assume that the messages are in *mh* format. The user must also specify which ruleset to use.

A message or folder of messages can be specified as either a file with a full pathname or as just the filename. If just the filename is specified, the path will be resolved according to the rules for that type of message. For example, *mh* uses the path returned by "mhpath"; whereas the current directory, or Unix environment search path is used for most other types). A ruleset can be specified as a file with a full pathname or as just the filename. Unresolved paths default to either the path set in the user configuration file (usually *~/.argus.rulesets* or to the current directory. The user can also specify whether or not to save user properties, message states, or alter data collection options.

Alinc

Alinc is a shell script, designed to make it easy for *mh* users to run rules over new mail. It takes the same arguments as ***runrules*** and then runs the specified ruleset on the specified folder. ***alinc inc***s mail into the inbox and then runs the specified ruleset. This is an example of how an Argus program can be called from a Unix shellscript to perform a useful function.

```
alinc  newmailrules +inbox
```

Multiple Rulesets

Users can create as many rulesets as they like. For example, a user might choose to have one set of rules for prioritizing new mail and another for sorting and filing old messages. It is also possible to create rulesets that are designed to be run from within other rulesets, as shown in Figure 6-2. Note that the *stop* action stops further processing of the current ruleset on the current message, but may continue with previously-called rulesets.

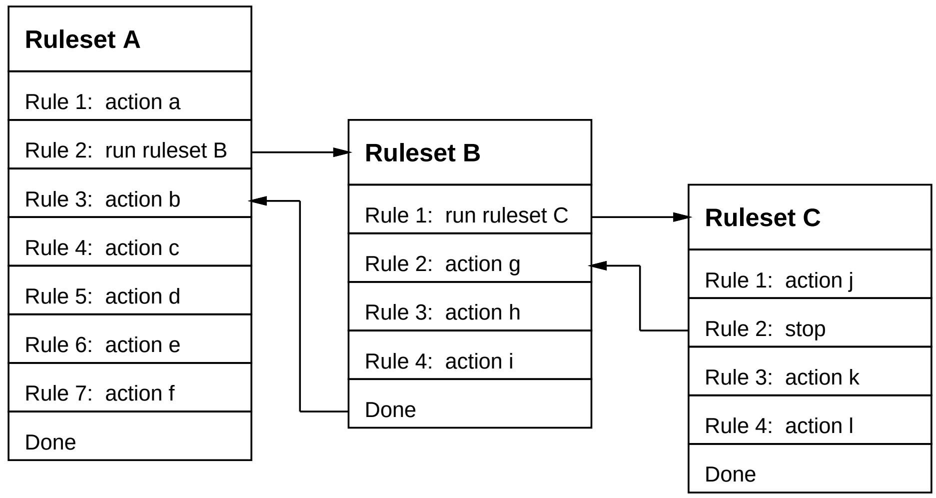

Figure 6-2: Multiple Rulesets:
Run Ruleset X runs another ruleset.
Stop prevents further firing in current ruleset.

7. Customizing Argus

Argus allows users to specify default values in several places. All settings have a system-wide default. Users can specify defaults in their personal directories, in the *~/.argus.profile*, which overrides system defaults. Individual commands can temporarily override these default values for the duration of the command.

The argus profile specifies default values for the message source, the the location of argus files, whether or not user properties and states will be saved and preferred orderings of menu alternatives. Refer to Appendix II for the current version of the Argus profile.

Argus creates a number of files and directories for users the first time they run any Argus program. (Users can change the name and locations of most of the files and directories by changing the *~/.argus.profile*.) Figure 7-1 shows how the user's files are organized and also shows the current organization of the Argus system files.

.argus.profile This contains non-standard defaults, such as the location of other Argus directories and files. This file is placed in the user's home directory.

argus A subdirectory of the user's home directory that contains user-specific Argus files.

argus/rulesets A subdirectory that contains the user's rulesets.

argus/data A subdirectory that contains data about the user's use of Argus.

argus.data A file in the *argus/data* subdirectory that keeps track of what data Argus should collect and for what time periods. It also holds temporary data for time periods that have not ended.

Data Collection

Argus can log statistics about its use. Argus can log each time a user creates, deletes, or modifies a rule or ruleset. It can keep track of the number of messages of different types it has processed over a given time period. It can keep track of how many times each rule was tested and fired over a given time period. The user can control what kind of logging Argus does, and what the time periods for logging are. Figure 7-2 shows the Argusi program, which allows users to control which data is collected and what time intervals to use.

Data is collected in two ways. In some situations, a complete record of every event is important. For example, every rule is stored and each change is recorded. In other situations, the rate of activity is of interest. In these situations, a starting time must be specified and the length of the interval during which information will be recorded. For example, every time a rule fires within a specified time period, a counter is incremented. The firing rate per specified interval is calculated at the end of the period. Argus allows users to specify any length period and either provide a specific starting time (e.g. record data at 2 am on Sundays) or from the time of initialization of the data collection file.

The types of data collected are:

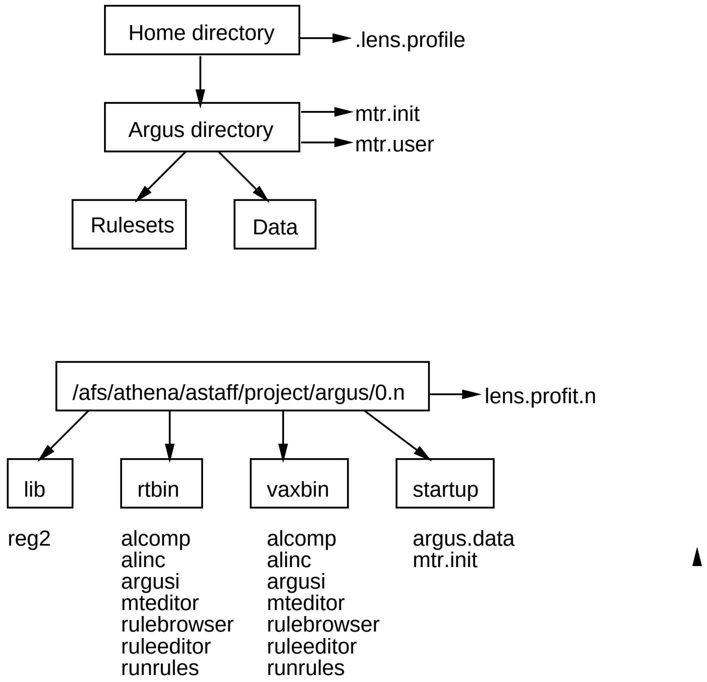

Figure 7-1: Argus Directories

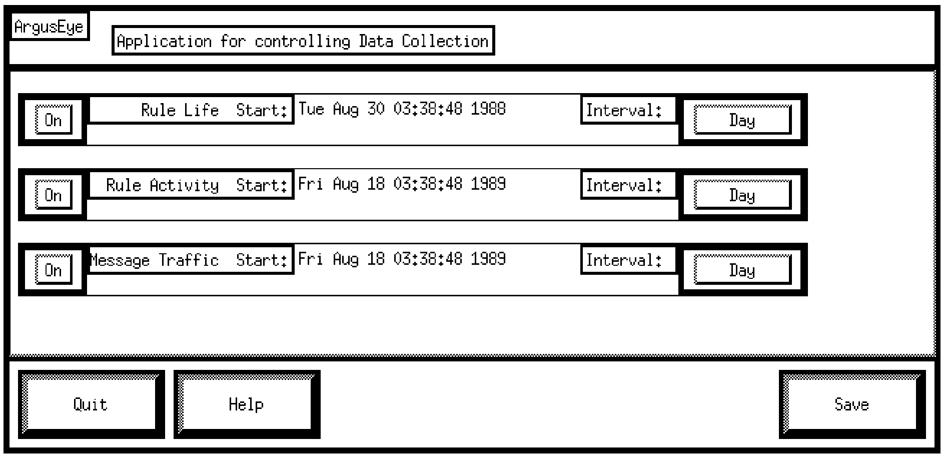

Figure 7-2: Argusi: User's Control Over Data Collection

1. **Message history:** Saves how many messages of each type have been processed by rules over a specified period of time. *msg.log*: stores data about messages that are processed by Argus, such as the *type* of the message and the time at which it was processed. Each time Argus handles a message the *msg.log* records the time of the event and the type of message processed.

2. **Rule activity:** Saves the firing history of each rule over a specified period of time. *ra.log*: stores information for every rule in each ruleset; the information shows how many messages a rule was tested on and, of those, how many the rule fired on.

   Unlike with rule life and message activity logging, information about rule activity is not immediately recorded in a log file. The data collection module keeps track of rule activity over a certain period of time (which the user modify). Then, at the end of the time period, statistics are recorded in *ra.log* for each rule in every ruleset for that period. The statistics show how many messages the rule was tested on and how many the rule fired on.

3. **Rule life:** Saves the history of how rules are edited and modified over time. *rl.log*: stores data about creation, changing, and deletion of rules and rulesets. Each time a rule or ruleset is created, deleted, or modified, the *rl.log* file records the event. A ruleset is considered to be modified when a rule is added to or removed from the ruleset, or when a rule in the ruleset is modified. The log file refers to rules and rulesets by the date at which they were last changed. Rulesets are also identified by their name, whereas for rules the names are optional.

4. **Snapshots:** Saves the state of the user's rules and message types at the time specified.

5. **Rule Firing Data:** Collected by the data collection module. Saves rule firing history, rule creation history, and snapshots of rules at specifiable periods of time.

8. Impact on Athena

The impact on Athena resources for the personal rule system is small. Individuals have the ability to create rules that duplicate messages in their own directories, but they must deal with the consequences of their actions.

The major area that would have had an impact on Athena is the general purpose rule server. The original version did not scale well, because it forced a large quantity of messages through the mail hub at once. A partial solution was tried that forced the number of messages to be sent slowly, but this would not work for very large numbers of messages. A better solution would be to integrate the rule system with the distributed network file exchange system with redundant servers. See: Cattey, French, Lewis extension below.

Argus has the potential to effect the following:

1. Disk storage: Data collection files may get large if left running for a long time. This depends on the level of use. One steady user has accumulated approximately 176k of data over a period of one year. Argus creates a set of directories in the user's path, including a ruleset area, data area, Xresources, and a profile. The same user has 205k of files (including data) in the argus directories.
2. Processing: Argus can take a long time to process a large volume of messages (still waiting for metrics). It is particularly slow with *mh*, because it uses *mh* commands and opens a new shell for each.
3. Network: The original design of the rule server has been changed to avoid flooding the mail hub. The proposed implementation will use the distributed file approach used by *turnin/pickup*, in a future release. Increasing the benefits of using electronic messages may have an indirect effect on network traffic.
4. Message type naming: The current set of message types can be updated by members of the Argus group. The message types used by Athena should be handed over to an Athena administrative group instead.

Argus has several dependencies. In particular, Argus draws on the formats of various electronic messages, which must be kept up-to-date. Argus also uses a program called *getdate*, which has a bug. The Argus code includes a work around. The Motif programs are all dependent upon the current state of the Motif widgets.

9. Proposed Extensions

Argus can be extended in three basic areas: the rule language, predicates, and actions. External changes, such as using a distributed server scheme to distribute rules, are also possible. The following are areas that could increase Argus's usefulness for both education and operations.

- Rule action registry: Different applications may require limiting the number of possible actions for rules. (Particularly if multiple users are creating rules about the same database of information.) Creating an action registry would provide this flexibility.

- Control Panel with rule status: Currently, all Argus applications can be called from the Unix command line (e.g. ruleeditor, runrules). We would like to have a single Motif application that allows users to run the different programs, change user preferences and data collection information, obtain status information and get help about the system.

- Rule language: The current rule language is quite simple. Rather than extending it, it would be better to make it accessible to a "real" rule language. It would be important to maintain the simple interface for general use, via the rule editor, but there are a number of applications that would benefit from a more sophisticated rule system. A simple extension would be the ability for rules to dynamically create rules, which would enable Argus to learn.

- Generalized translation module: It would be useful to provide a simple system for teaching Argus how to translate new types of messages into the standard message format. The module must also be able to keep mailer-specific properties available. Currently, we would like to add: zephyr, discuss, OLC, turnin-pickup and general streams of system messages.

- Intermessage processing of rules: Each rule in a ruleset has access to each message in turn. Therefore, the only information available to the predicate of a rule is the information available from the current message. There is no inter-rule communication. This presents problems when one wishes to compare two different messages, as in a sorting algorithm. Several solutions are possible. One is to extend the current data collection system, which does collect information about groups of messages (and stores the results in a message format which can be processed by rules). Another solution is to add variables associated with a ruleset. This would allow a rule to set a 'global' variable to be used by another rule in the same ruleset.

- Access to relative position of a message in a stream: This would be useful for cleanup operations when the last message in a stream is being processed. This would also be useful in order to keep track of the number of messages which have been processed so far. This would be useful, for example, to keep track of the amount of storage space used so far. The general problem of accessing the relative position of a message in a stream leads us to consider the possibilities of pre and post processing of rulesets/folders. For example, when filtering has been completed on a folder the user may then want to send the contents of the folder to a printer for a hardcopy.

- Message types: There is still no robust mechanism to handle message types within and across organizational boundaries. The current message typing scheme is informal and must be handled by people. (Note: this may be the only way to handle such things.)

- Action indicator: Provide information to the user about which actions have been taken by the ruleset, e.g. that 10 messages have been moved to a particular folder. The information can be computed from the data collected, but is not immediately accessible.

- Comparing the predicates of different message components: The predicates of Argus do not allow the comparison of components to other components. That is, the user cannot create a predicate that determines if the CC: field is equivalent to the FROM: field.

- Distributed rule system: Originally conceived of as the "anyone server" in the Information Lens, this has always had serious scaling problems, both from the user's perspective and from the system resource perspective. The idea is that individuals send rulesets to a central location and large groups of messages are passed through each ruleset. As the rulesets increase in size and number, the system slows down. Also, the traffic on the network increases. Cattey, French and Lewis have proposed a distributed network file exchange system with redundant servers to handle course assignments, through *turnin/pickup*. The Argus rule editor could provide a more sophisticated front end to their system, and their system would provide the necessary backend to handle the scaling problems. Figure 9-1 illustrates how the Argus system could interface to the proposed extension to *turnin/pickup*.

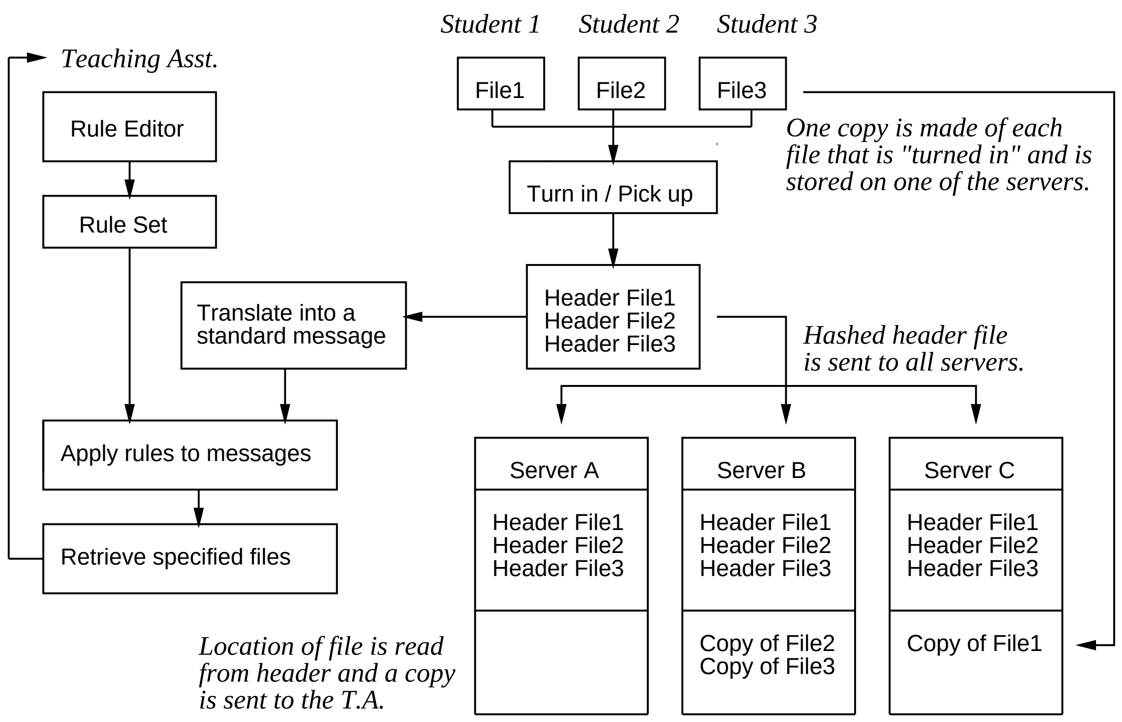

Figure 9-1: Proposed relationship between Turnin/Pickup and Argus

I. Glossary

Rule	An IF predicate, consisting of one or more typed components, and a THEN section, consisting of one or more actions. Rules are created by the ruleeditor and stored as ASCII text.
Message	A set of fields, with an optional unstructured section, that can be compared to Argus rules.
Folder	An internal representation of a collection of messages. Can be an open-ended stream or a close-ended set.
Translation Module	Translates externally-created messages into single, internally understood format. Keeps information specific to the message in a source-specific property.
Ruleset	A named collection of rules, stored as ASCII text in a file.
Message source	The source of a message, external to Argus. Sources could include: *mh*, *rmail*, *zephyr*, *discuss*, or *turnin/pickup*.
Rule server	A server that accepts both rules and messages. It runs rulesets over specified messages.
Message type	A pre-defined collection of fields in a message. A phone message might contain the fields: TO:, FROM:, CALLER:, PHONE NUMBER: and ACTION:.
mh	The Rand Mail Handler. Mail messages are stored in separate files in user directories. Access is via shell commands.
rmail	Mail system, accessible from emacs. Uses babyl mail format. Mail messages are stored in a single file.
zephyr	Athena's notice and transport message delivery system. Messages are not stored.
discuss	Athena's networked conferencing system. Messages are stored in "meetings" and may refer to other messages.
turnin/pickup	Athena's system for allowing students to turn in assignments to TA's or Professors, and pick up graded versions.
rule browser	Argus program. Allows users to view, print, rearrange, edit, add, delete, and copy rules in a ruleset.
registry	Internal argus function that provides a consistent interface for associating named pairs. Includes: message type registry, ruleset registry, action registry.
darray	Internal argus function that provides a consistent interface for storing internal data structures.
message traffic	Data collected about how often messages of each type are processed by which rules in which ruleset over a user-defined period of time.
rule activity	Data collected about how often rules are tested and fired in each ruleset over a user-defined period of time.
rule life	Data collected about additions, changes and deletions of rules.
predicate	The "IF" part of a rule. May evaluate one or more attributes of a message.
action	When the predicate of a rule is evaluated as "true", the "Then" part of a rule may perform one or more actions. Actions consist of a verb and a variable number of arguments.
attribute	The parts of a message tested by the "IF" part of a rule. Can included fields, properties or states.
field	An attribute of a message. The physical contents of a message, of the form: <field>: <value>. May be supplied by the sender, by the sender's message system, by the reciever's message system, by a rule action, or by the receiver through use of a text

	editor or other annotation system.
property	An attribute of a message. Three types: defined by user, supplied by originating message system and calculated about the message.
state	An attribute of a message. The state of processing of the current ruleset.
rule stream	ruleset or collection of rules.
message stream	folder or collection of messages.

II. Lens Profile

Note: The references to "lens" should be replaced with "argus".

Note: This is a copy of the profile in /afs/athena/astaff/project/argus/lens.profile.0.5.

```
#
# The default for every profile component should be specified
# in this file
#
MailUserAgent:
        mh
LensDirectory:
        $(HOME)/lens
RulesetDirectory:
        $(HOME)/lens/rulesets
ActionDirectory:
        $(HOME)/lens/actions
MessageTypeRegistryInit:
        $(HOME)/lens/mtr.init
SaveStates:
        False
SaveProperties:
        True
DataDirectory:
        $(HOME)/lens/data
DataStateFile:
        $(HOME)/lens/data/lens.data
StartupDataStateFile:
        /afs/athena/mit/argus/0.5/startup/lens.data
MsgDataOutput:
        $(HOME)/lens/data/msg.log
RuleActivityOutput:
        $(HOME)/lens/data/lens.ra.log
RuleLifeOutput:
        $(HOME)/lens/data/lens.rl.log
StartupMessageTypeRegistryInit:
        /afs/athena/mit/argus/0.5/startup/mtr.init
SnapShotOutput:
        $(HOME)/lens/data/lens.data.snap
DefaultRuleMessageType:
        MSG
#Not currently implemented
DefaultSendMessageType:
        MSG
VerbPresentationOrder:
        move-to,copy-to,delete,stop,runrules,shell,set,clear,add,
        subtract,nonpermanent,permanent,remail,sendme,system,
        systembody
FieldTypePresentationOrder:
        Any-Case-Text,Same-Case-Text,Date,Boolean,Integer,
        Any-Case-Text-with-commas-as-ORs,
        Same-Case-Text-with-commas-as-ORs
```

```
NextProfilePath:
        $(HOME)/.lens.profile
DefaultFieldType:
        Any-Case-Text-with-commas-as-ORs
AnyoneDropDirectory:
        /afs/athena/mit/argus/athena-anyone/drop
AnyoneLibDirectory:
        /afs/athena/mit/argus/athena-anyone/lib
AnyoneName:
        athena-anyone
DefaultRuleVerbs:
        move-to
DefaultAnyoneServer:
        athena-anyone
SystemMessageTypeRegistry:
        /afs/athena/mit/argus/0.5/lib/reg2
```

III. Sample Ruleset

```
$format_version 1
$desc <no description>
$created 617153873
$updated 617155560

$created 617155393
$updated 617155393
$name r1
$if
  "1-week-remind:" "%is_before" "1 week ago" "Date" %0
 %and 1
$then
 "reminder" %copy-to 1
 "zwrite" %system 1
$end

$created 617155393
$updated 617155393
$name d1
$if
  "1-week-delete:" "%is_before" "1 week ago" "Date" %0
 %and 1
$then
 %delete 0
$end

$created 617343086
$updated 617343086
$name argus
$if
   "To:" "%contains" "argus" "Any-Case-Text" %0
   "cc:" "%contains" "argus" "Any-Case-Text" %0
  %or 2
 %and 1
$then
 "siftargus" %move-to 1
$end

$created 608015888
$updated 608015888
$name From me
$if
  "From:" "%contains" "mackay" "Any-Case-Text" %0
 %and 1
$then
 "siftme" %move-to 1
$end
```

```
$created 616781895
$updated 616781895
$name Mailer daemon
$if
  "From:" "%contains" "mailer-daemon" "Any-Case-Text" %0
 %and 1
$then
 "siftdemon" %move-to 1
$end

$created 617656115
$updated 617656115
$name <no name>
$if
   "To:" "%contains" "argus" "Any-Case-Text" %0
   "cc:" "%contains" "argus" "Any-Case-Text" %0
  %or 2
 %and 1
$then
 "siftargus" %move-to 1
$end

$created 618959256
$updated 618959256
$name <no name>
$if
   "To:" "%contains" "staff@athena" "Any-Case-Text" %0
   "cc:" "%contains" "staff@athena" "Any-Case-Text" %0
  %or 2
 %and 1
$then
 "siftstaff" %move-to 1
$end
```

IV. Message Type Registry

```
#MT: 8
NAME: bug-report
PARENT NAME: MSG
FIELDS: {
        To: <Any-Case-Text>
        From: <Any-Case-Text>
        Subject: <Any-Case-Text>
        Date: <Date>
        Software: <Any-Case-Text>
        Bug: <Any-Case-Text>
        Severity: <Any-Case-Text>
}

NAME: meeting
PARENT NAME: MSG
FIELDS: {
        To: <Any-Case-Text>
        From: <Any-Case-Text>
        cc: <Any-Case-Text>
        Subject: <Any-Case-Text>
        place: <Any-Case-Text>
        time: <Date>
}

NAME: feature-request
PARENT NAME: MSG
FIELDS: {
        To: <Any-Case-Text>
        From: <Any-Case-Text>
        Subject: <Any-Case-Text>
        Date: <Date>
        Argus-application: <Any-Case-Text>
        Feature-requested: <Any-Case-Text>
        Severity-of-need-1..10: <Integer>
}

NAME: feature-announcement
PARENT NAME:
FIELDS: {
        To: <Any-Case-Text>
        Cc: <Any-Case-Text>
        Argus-application: <Any-Case-Text>
        New-feature: <Any-Case-Text>
}
```

```
NAME: phone
PARENT NAME: MSG
FIELDS: {
        To: <Any-Case-Text>
        From: <Any-Case-Text>
        cc: <Any-Case-Text>
        Subject: <Any-Case-Text>
        Caller: <Any-Case-Text>
        when: <Date>
        about: <Any-Case-Text>
        return-call: <Any-Case-Text>
}

NAME: MSG
PARENT NAME:
FIELDS: {
        To: <Any-Case-Text>
        From: <Any-Case-Text>
        Subject: <Any-Case-Text>
        Date: <Date>
}

NAME: rule
PARENT NAME:
FIELDS: {
        To: <Any-Case-Text-with-commas-as-ORs>
        Subject: <Any-Case-Text-with-commas-as-ORs>
}

NAME: bug-fix
PARENT NAME: MSG
FIELDS: {
        To: <Any-Case-Text>
        From: <Any-Case-Text>
        Subject: <Any-Case-Text>
        Date: <Date>
        Software: <Any-Case-Text>
        Old-bug: <Any-Case-Text>
        The-fix: <Any-Case-Text>
}
```

V. Data Collection Files

Rule Activity Log

```
History between Fri Jul 14 03:38:48 1989 and Sat Jul 15 03:38:48 1989
Rule Wed Apr  5 19:03:48 1989
        In RuleSet sifter
                Fired 0 out of 253 times
                on Message Type unknown
                Fired 0 out of 1 times
                on Message Type MSG
                Fired 0 out of 2 times
                on Message Type feature-announcement
Rule Wed Apr  5 19:02:27 1989
        In RuleSet sifter
                Fired 0 out of 253 times
                on Message Type unknown
                Fired 0 out of 1 times
                on Message Type MSG
                Fired 0 out of 2 times
                on Message Type feature-announcement
Rule Wed Apr  5 19:07:16 1989
        In RuleSet sifter
                Fired 2 out of 253 times
                on Message Type unknown
                Fired 0 out of 1 times
                on Message Type MSG
                Fired 0 out of 2 times
                on Message Type feature-announcement
Rule Wed Apr  5 19:04:52 1989
        In RuleSet sifter
                Fired 9 out of 253 times
                on Message Type unknown
                Fired 1 out of 1 times
                on Message Type MSG
                Fired 2 out of 2 times
                on Message Type feature-announcement
```

Proposed format change:

```
Rulename: Wed Apr 5 19:04:52 1989
RuleSet:  sifter

Fired: 9
Outof: 253
Message type:  unknown

Fired: 1
Outof: 1
Message Type: MSG

Fired: 2
Outof: 2
Message Type: feature-announcement
```

Message Traffic Log

```
History between Fri Jul 14 03:38:48 1989 and Sat Jul 15 03:38:48 1989
Number of messages of type feature-announcement
        shuffled or unspecified (unknown): 1
        received (incoming): 0
        sent (outgoing): 0
Number of messages of type MSG
        shuffled or unspecified (unknown): 1
        received (incoming): 0
        sent (outgoing): 0
Number of messages of type unknown
        shuffled or unspecified (unknown): 253
        received (incoming): 0
        sent (outgoing): 0
Number of messages of type feature-announcement
        shuffled or unspecified (unknown): 1
        received (incoming): 0
        sent (outgoing): 0

History between Sat Jul 15 03:38:48 1989 and Sun Jul 16 03:38:48 1989
Number of messages of type unknown
        shuffled or unspecified (unknown): 196
        received (incoming): 0
        sent (outgoing): 0
```

Proposed format change:

```
Start History: Sat Jul 15 03:38:48 1989
End History:   Sun Jul 16 03:38:48 1989
Message type:  unknown
Unspecified: 196
Incoming: 0
Outgoing: 0
```

Rule Life Log

```
RuleSet modified: "newmail": (was Tue Aug 30 12:04:06 1988) Tue
Aug 30 12:51:35 1989
        Rule deleted (oldphonemessage) Tue Aug 30 12:04:05 1989

RuleSet created: "sendmail": Tue Aug 30 12:56:58 1989

Rule Created: () ID: Tue Aug 30 13:07:13 1989
If: (and [To word= ] [cc word= ] [From word= ] [Subject word= ]
[Caller word= ] [when word= ] [about word= ] [return-call word= ] )
Then: (system zwrite mackay You have a phone call)

RuleSet modified: "sendmail": (was Tue Aug 30 13:04:40 1989)
Tue Aug 30 13:07:14 1989
        Rule added () Tue Aug 30 13:07:13 1989

Rule Created: () ID: Tue Aug 30 13:13:41 1989
If: (and [To word=  ] [cc word= ] [From word= ] [Subject word= phone]
[Caller word= ] [when word= ] [about word= ] [return-call word= ] )
Then: (system zwrite mackay You have a phone call)
```

Proposed format change:

```
Rule name:  Tue Aug 30 13:13:41 1989
Activity:   Created:
If: (and [To word=  ] [cc word= ] [From word= ] [Subject word= phone]
    [Caller word= ] [when word= ] [about word= ] [return-call word= ])
Then: (system zwrite mackay You have a phone call)
```

VI. Lens data collection file: lens.data

Annotated version of the lens.data file

The top part of the file contains information received from Arguseye.

```
Format_version 6
**RULE-LIFE:        Data on creation, modification, deletion of rules
1                       Enable switch:  1 = on  0 = off
0                       Currently within the time period? 1=yes, 0=no
620120328               Start time for data collection
620123928               Stop time for data collection
3600                    Data collection interval (derived from above)
**RULE-ACTV:        Data on when rules fire on what messages
1                       Enable switch:  1 = on  0 = off
1                       Currently within the time period? 1=yes, 0=no
620984328               Start time for data collection
621070728               Stop time for data collection
86400                   Data collection interval (derived from above)
**MSG:              Data on types of messages processed & rules fired
1                       Enable switch:  1 = on  0 = off
0                       Currently within the time period? 1=yes, 0=no
620984328               Start time for data collection
621070728               Stop time for data collection
86400                   Data collection interval (derived from above)
```

The bottom contains temporary information being stored for subsequent compression by message traffic and rule activity data files. (Rule life does not have temporary data.)

Rule activity temporary data:

```
0: 10                   0 indicates Rule activity data.
                            10 rules follow.
617342960 #1            Rulename not specified so used creation date.
                            1 ruleset follows.
sifter #1               Ruleset name is sifter
                            1 rule follows
unknown 109 0 0 0 0 0   Message type is "unknown" (should be "MSG")
                          3 pairs indicating total msgs/total firings
                          pair 1: unspecified.  109 messages. 0 fired.
                          pair 2: incoming.     0 messages.   0 fired.
                          pair 3: outgoing.     0 messages.   0 fired.
616782261 #1            Next rule to be analyzed, 1 ruleset follows.
sifter #1               Ruleset name is sifter, 1 rule follows
unknown 109 0 0 0 0 0   etc.
617343154 #1
```

Message traffic temporary data:

```
1: 2                    1 indicates message traffic data.
                            2 messages follow.
unknown 49 0 0 0 0 0    Message type is "unknown" (should be "MSG")
                            3 pairs. 1st is message.  2nd always 0.
                                pair 1: unspecified.  49 messages.
                                pair 2: incoming.      0 messages.
                                pair 3: outgoing.      0 messages.
unknown 60 0 0 0 0 0    Next message to be analyzed.
```

VII. Argus Command Line Specification

Sample format for the command line usage message:

```
Usage:     rulebrowser [-help] [-new] ruleset

arguments:
    ruleset   name of the ruleset

switches:
    -h
    -help     Lists this documentation
    -n
    -new      Specifies that a new ruleset is to be created

examples:
    rulebrowser myrules
    rulebrowser -n mynewrules

for help:   rulebrowser -help
```

Standard names for command line arguments

Structure	Description
filename	Name of file
folder	Name of a message folder
msgsrc	Source of a message (*mh*, *rmail*)
msgtype	Type of message (*bug-fix*, *phone*)
ruleset	Named group of rules
server	Name of a rule server
rulenum	Position of a rule in a ruleset

Switches that do not take arguments

Switch		Description
-e	-elim	Eliminate (delete)
-f	-force	Force action, don’t ask
-h	-help	Display usage message
-n	-new	Create new item (e.g. rule)

Switches with fixed arguments

Switch		Valid Args	Description
-a	-after	[rulenum]	Insert after specified rule number
-b	-before	[rulenum]	Insert before specified rule number
-d	-data	[all | none]	Turn on all data collection?
		[msg | nomsg]	Turn on message traffic data?
		[rl | norl]	Turn on rule life data?
		[ra | nora]	Turn on rule activity data?
-r	-rule	[rulenum]	Which rule number?
-s	-state	[on | off]	Save rule-processing state information?
-u	-uprop	[on | off]	Save user-defined properties?
-w	-warn	[on | off]	Verbose warning messages?

Switches with names as arguments

Switch	Description
-FNfilename	Filename for input/output
-FOfolder	Folder name
-MTmsgtype	Existing message type
-NMmsgtype	New message type
-LFfilename	Loading file
-RSruleset	Ruleset
-MSmsgsrc	Message Source
-SVserver	Rule server

Table of Contents

List of Figures