A Proposal for Creating a Motif Version of Argus

11 July 1989 at 17:02

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

Motivation

Project Athena creates and processes a large volume of electronic messages from a variety of sources, including mh, rmail, discuss, BBoard, and Zephyr. The purpose of Argus is to provide a simple, consistent and powerful way to manage these messages. Argus allows users to create if-then rules that look for specified characteristics of individual messages and perform one or more actions accordingly.

Argus takes advantage of the *semi-structure*[1] inherent in these messages. For example, a value that appears after the word **To:** in an electronic mail message is likely to be a user's name or a distribution list. Users may also create rules that operate on factors other than the content of a message. Some information about the message may be calculated, such as the message's ength. Other information may be provided by the sender, such as the inclusion of an additional field or the use of a pre-defined message type. The message recipient can also define properties of messages, either by marking them while reading messages or by creating rules that identify such properties and mark the messages accordingly.

Argus rules use this information to perform different actions. Rules can operate on the messages themselves (e.g. to sort, modify, delete or redistribute messages) or to perform other tasks (e.g. run other rules, generate new messages or call Unix programs). Rules are grouped into rulesets, which are applied to each message in turn. A user may run a ruleset over a specified group of messages at any time. Rulesets may also be triggered automatically by changes in the environment.

Argus is not intended to be a message system. Rather, it provides flexible annotation and rule-based processing of messages, regardless of their source or storage mechanism.

Examples

The following examples illustrate how different groups at Athena could use Argus to solve different kinds of problems.

- **Prioritizing and organizing electronic mail messages:**

  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 archived 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 "next-meeting" field, which can then be extracted and processed automatically. The principles here can be applied to Zephyr messages and extraction of information from discuss meetings.

- **Processing of automatically generated messages:**

[1] These messages are referred to as *semi-structured* messages because they are each composed of a structured section (e.g. the header field in a mail message) and an unstructured section (e.g. the body of the message).

Servers, printers, and other devices generate system status messages A system administrator could have an Argus application running on other machines on the network that would react to certain types of status messages. For example, a machine running an Argus application that receives a "down" message from a server could take over some services provided by that server. It could then automatically send out its own message across the network informing other machines that it can provide those services. Alternatively, a system administrator might wish to have some on-the-fly control in handling these messages, and could run her own rulesets on the status messages. She could decide on which rulesets to run based on information that might be outside of a fully automatic system's range (such as abnormally high system use, or air-conditioner failure in one machine room).

- **Rule Server:**

  A Rule Server acts as an intermediary between message senders and receivers; it annotates and redistributes messages. The Rule Server is another way a system administrator could use Argus to handle automatically generated system messages. For example, the administrator could place rules concerning status messages in one centralized location. This would make modifying rules concerning system message an easier task.

  As well as handling automatically generated messages, the Rule Server could handle personal messages sent to a specialized group. The group could use Argus to annotate and redistribute these messages to its (the group's) members. For example, an On Line Teaching Assistant (OLTA) system implemented with Argus might have a rule that annotates messages with a field indicating which TA should receive the message. The annotation might be based on the subject of the message and/or the time of day the Rule Server received the message. The Rule Server would then redistribute the messages to the propper TA's. A professer could easily change the redistribution choices since the rules would be centralized.

Existing System

A working prototype of Argus already exists. It supports a flexible and expandable rule language to which users have access through a window interface. Rules are stored in human readable form in *ruleset* files. A ruleset is a collection of one or more rules. The ruleset files use a post-fix notation to represent the rules.

Rules consist of a set of *IF* conditions and a set of *THEN* actions. The *IF* conditions operate on the structured fields within the message, calculated information about the message (e.g. length), and user-defined properties. Any number of these entities can be combined with boolean operators in the *IF* section. Argus executes the actions in the *THEN* section if the *IF* condition holds true.

Argus can perform a wide range of actions. For example, an action could invoke another ruleset on the message being tested, or it could invoke a shellscript or a program, such as *zwrite*. The range of actions can vary widely; they could become very complex or remain simple, depending on the users' tasks. Actions can also add user-customized *user-properties* and *states* to a message. This allows the users to put information relevant to their task into a message, which could aid them in further processing/sorting of the message through Argus, or in interpreting the message at a later date. The rich nature of the actions allows Argus users to create a virtually limitless set of powerful customized applications that deal with the transfer of information.

Argus translates messages into a uniform internal format for subsequent precessing. A seperate *folder* object interface must be created for each new kind of message. Currently folders exist that interface with RMAIL, mh, and xmh message formats. New kinds of messages, such as Zephyr or discuss, can be added by creating a seperate folder.

The existing modules that make up Argus are the following:

- **The Rule Editor:** gives users access to a subset of functions supported by the rule language. A user fills in fields in the *IF* section with target values that Argus will test against corresponding fields in messages. The user then fills in the *THEN* section with at least one action.
- **The Ruleset Editor:** allows users to look a set of rules and choose which ones to edit, move, copy or delete.
- **The Data Collection Controller:** through which users instruct Argus to log statistics and histories of their Argus 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.
- **The Message Type Editor:** allows users to create new message types. Message types are stored in personal or shared registries on disk.
- **AIinc:** allows users to run a specified ruleset prior to "inc'ing" new mail.
- **AIcomp:** allows users to create a message using a specified message type. AIcomp includes a message type and header fields in an Emacs buffer.
- **Runrules:** allows users to invoke ("run") rulesets on messages. The user specifies the messages by specifying their *folder*. A *folder* is any container for messages that Argus knows about. An "RMAIL" file is a folder, as is an "inbox" directory. The user must specify which folder format Argus should use, otherwise Argus uses the "mh" format.

Argus will create a number of files and directories for users the first time they run any Argus module; users can modify these files. These are the standard default directories and files:

- .lens.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.
- lens: A subdirectory of the user's home directory that contains user-specific Argus files.
- lens/rulesets: A subdirectory that contains the user's rulesets.
- lens/data: A subdirectory that contains data about the user's use of Argus.
- lens.data: A file in the lens/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.

Users can change the name and locations of most of the files and directories by placing the new names and locations in *.lens.profile*.

Proposal for Continuing Work

Although a basic Argus system exists, changes and improvements need to be made:

- Currently all code for user interface via windows uses the Athena and HP widget sets. We would like to convert Argus to use the Motif widgets in order to be compatible with Athena's imminent Motif environment. We would also like to incorporate results of our user interface study by redesigning some of Argus. During the past few months we have received feedback from users about some weak points and awkward characteristics of Argus' operation. We would like the opportunity to rework certain areas of Argus to eradicate these problems. Much of this reworking could be contiguous with the conversion to Motif since much of it is concerned with user interface related issues.

- As its developers, we have had the chance to analyze Argus from a fundamental conceptual and structural perspective. We would like to make functional enhancements to certain areas of Argus that would require reworking some underlying components. One of the components that we believe needs enhancements and restructuring is the rule language.

- Currently Argus has interfaces to RMAIL, mh, and xmh, and can receive messages via these three sources. Systems maintenance and administration groups would benefit from an interface with the *Zephyr Notification Service*. This is one of the new components we wish to build in order to expand Argus' functional domain. Other enhancements simply require adding extensions to existing components. Some of these relate to the rule, ruleset, and message type editors, and can be implemented during the conversion to Motif.

- Currently the Rule Server sends acutal copies of a given message when it redistributes. To prevent the needless propagation of messages we would like to make changes so that the Rule Server sends only "pointers" to messages to users, rather than an actual copy. This would require more than trivial changes, but would make the Rule Server more practical for high volume message traffic.

Dan’s fun-key system stuff

============= Proposal 1:

Demonstration project for LENS with direct application to Athena development and operational needs.

Problem Statement:

At the Athena design point of 10,000 hosts, maintenance and management will require prospective tools, not merely the reactionary ones employed today. Part of the solution is to gather all the available information from hosts in the field into a central location. This gathering would result in a high volume of information. That high volume of information would need to be sifted to be of significant and timely value.

Constraints:

- Must not impede the collection rate of information.
- Must be consistent with other aspects of the Athena service model. Examples include authentication and name to service location lookup.
- Idempotent transformation of the data stream.
- Able to count events, because time rates of events is often the information of interest.
- Able to raise alarms at relevant and controllable levels, including use of all aspects of the Athena communication plant.

Schedule:

Desirable for it to be in place by 1 September 89.

Desirable for workstation mods, if any, to collect data in format acceptable to LENS to be in place by 1 June 89.

============= Proposal 2 Summary:

Second demonstration project for LENS with direct application to Athena development and operational needs.

Problem Statement:

The process of preparing an Athena software release in its fullest flower involves many hours, some 1300 files containing compilation instructions, and many thousands of lines of logging information. This logging information is necessarily a haystack; we need to find the needles in it. The needles are the events in the compilation process that are important, where important is heuristic. Makelogs seem to meet the loose structure requirements of lens.

Constraints:

Must be sensitive (detecting true positives) and specific (detecting true negatives) and have high predictive value (probability of an event being truly interesting must be high among events tagged as potentially interesting).

Why an item was selected must be part of the problem report, as must precise in-log location information.

Idempotent transformation of the data stream.

Able to raise alarms at relevant and controllable levels, including use of all aspects of the Athena communication plant.

Schedule:

Desirable for it to be in place by 1 June 89.

==============

Proposal 3 Summary:

A third demonstration project for LENS with direct application to Athena development and operational needs.

Problem Statement:

The on-line consulting system (OLC) is a rendevous mechanism for persons with questions and persons with answers. The questions and their answers are logged both for quality control purposes and for study. The quality control questions can be answered at the gross level with process type information derivable from the logs, time to query terminations, peak rates, distributions of queries by workstation and or user characteristics. The study purposes involve detection of similar questions, so as to populate a ‘‘stock answer’’ section of the service, and to evaluate extension of the service into course consulting directly (utilizing TA’s as course-specific consultants).

Constraints:

Must be able to operate on existing logs, possibly to be trained thereon.

Idempotent transformation of the data stream.

Able to count events, because time rates of events is often the information of interest.

Able to iteratively reclassify and to multiply classify particular events.

Privacy controls are important and must be reflected in the nature of the reporting, i.e. analysis must be reportable in noninterventionist style.

Schedule:

Desirable for it to be in place by 1 September 89.

Desirable for workstation mods, if any, to collect data in format acceptable to LENS to be in place by 1 June 89.

Please advise.

--dan

Table of Contents