The Architecture of MAITA
A Tool For Monitoring, Analysis, and Interpretation

Jon Doyle,[1] **Isaac Kohane,**[2] **William Long,**[1] **Peter Szolovits**[1]

[1] Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139
[2] Childrens' Hospital, Boston, MA 02139

Abstract: This report describes the aims, functions, and organization of the MAITA system for knowledge-based construction, adaptation, and control of networks of monitoring processes.

DRAFT

Version of September 21, 1999 at 12 : 24

Comments are welcome. Please address comments to doyle@mit.edu.

This document is available via
http://www.medg.lcs.mit.edu/projects/maita/.

Acknowledgments

We thank several people for their help in the MAITA project.

- Philip Greenspun suggested the notion of a database-backed monitor of monitors and helped with TCL, AOLServer, and Oracle in our initial implementations.
- Cungen Cao prepared the first major implementation of the MOM and sample monitoring processes in Java.
- Mary Desouza helped with the library knowledge base design.
- Neil MacIntosh provided NICU data sets used in some of our demonstrations.
- Christine Tsien provided useful advice on monitoring problems.

The initial versions of MAITA monitoring systems were implemented using Allegro Common Lisp, TCL, AOLServer, and Oracle. We thank Franz corporation with help in the Lisp components.

Funding for this work was provided by the Defense Advanced Research Projects Agency, first through the High Performance Knowledge Bases (HPKB) program (grant F30602-97-1-0193), and subsequently through the Information Assurance (IA) and Cyber Command and Control (CC2) programs (grant F30602-99-1-0509). Expression of views herein does not imply endorsement by DARPA.

Contents

Chapter 1

Introduction

The MAITA system provides knowledge-based support for monitoring tasks. The acronym stands for the *Monitoring, Analysis, and Interpretation Tool Arsenal.* The name reflects the provision of tools rather than monitoring systems themselves. The hypothesis underlying this work is that one can significantly reduce the costs—in time, effort and expertise—of constructing a monitoring system through use of a rich library of monitoring systems and tools that enable easy composition, modification, testing and abstraction of these library elements. The arsenal thus includes tools for solving monitoring problems, tools which permit composition and correlation of separate signals into informed alerts, and tools which permit rapid addition or tailoring of monitor behavior.

This work was undertaken to construct a uniform monitoring infrastructure of use to multiple investigations conducted by the Clinical Decision Making group at the Massachusetts Institute of Technology Laboratory for Computer Science and the Informatics Program at Childrens' Hospital of Boston. We aim to use the many monitoring needs of these multiple investigations as a source of tasks through which to test the hypothesis.

This report seeks to describe, explain, and specify the MAITA system and its implementation. To simplify the description in the face of an ongoing implementation effort, we use the present tense to refer to aspects of the expected implementation even though some of the elements of the system described here have not been implemented. We expect a fairly complete implementation by the spring of 2000.

1.1 Motivation

We designed the MAITA system to help address two difficult problems: monitoring in context, and monitoring of special or temporary concerns.

Hospital intensive care units (ICUs) provide good examples of the problem of monitoring in context. Each ICU has dozens of monitoring devices attached to the patient, and each device signals alerts if the one or several signals it measures oversteps certain (usually narrow) bounds. Since even simple movements of the patient can cause momentary loss of signal, this herd of dissociated monitors typically sounds an alarm once or twice every minute. Investigating and disposing of each alert takes a trained nurse or physician several minutes, so attendants frequently disable most or all of the alarms to allow themselves to attend to the patient rather than to the machines. Though disabling the alarms defeats the purpose of the alarms, it does not always have the terrible consequences one might imagine because most of the alerts are false alerts, ones which can be seen to be false right away with some knowledge of medicine. For example, a heart rate of zero in the presence of steady blood pressure, blood oxygen, and brain activity signals a detached heart rate probe, not a stopped heart. The problem here is to find a way to use medical knowledge and common sense to combine all the hundreds of signals into considered alerts that represent the true conditions requiring attention of the attendants.

Computer security provides good examples of the problem of special or temporary concerns. Miscreants attempting to break into one site often repeat their attacks at other sites. A successful or partially successful attack against one site may call for alerting similar or related sites of the special characteristics of the attack, since the attack may have involved several events which mean little in themselves but which in retrospect appeared essential elements of the successful attack. For example, a random request from an obscure Lilliputian site might immediately precede a seemingly unrelated request from a Blefescucian site in the course of the attack on a particular service. If the succumbing site can alert its neighbors to this fact, the neighbors may repel similar attacks by watching for this specific combination and taking appropriate ameliatory actions pending a longer term solution to the vulnerability. The problem here is how to rapidly modify monitoring systems in place to recognize additional or specialized conditions and how to easily remove these specialized additions as windows of opportunity close or shift.

1.2 Overview

The MAITA system provides tools both for constructing intelligent monitoring networks that exploit domain knowledge to sift and correlate source-level signals, and for rapidly modifying running networks to address specialized or temporary concerns.

Chapter 2 sketches the main monitoring concepts involved in the MAITA system. Several subsequent chapters cover the main system components. Chapter 3 details an architecture for networks of distributed monitoring processes and their communications. Chapter 4 explains the architecture of the command and control system used to operate distributed monitoring networks.

This document does not describe the MAITA language for describing temporal signals, patterns, events, and monitoring processes, which is still under development. The remainder of this overview outlines the motivations for the overall system organization.

1.2.1 Distributed monitoring

The central concept in the MAITA architecture is that of a network of distributed monitoring processes. The metaphor we use for thinking of the operation of these monitoring networks is that of electrical networks, in which we “wire together” various components and network fragments by connecting their terminals together. In the computational context, individual monitoring processes take the place of electrical components, and transmitting streams of reports takes the place of electrical conduction. The set of monitoring processes form the nodes of the network, and the communication paths form the edges or links of the network. Each process in the network may have a number of “terminals”, each of which receives or emits streams of reports. The network may exhibit a hierarchical structure, as some monitoring processes may consist of a subnetwork of subprocesses.

1.2.2 Stable monitoring

The distributed processes may degenerate into chaotic interference without some means for structuring the interactions. To provide this structure, MAITA provides a“monitor of monitors” or “MOM” to construct, maintain, inspect, and modify the monitoring network and its operation. We achieve a degree of uniformity in the control process by organizing MOMs as special types of monitoring processes.

The MOM is designed to provide for resilient and persistent networks of monitoring processes. Toward this end, the command and control system monitors all the other monitoring processes, correcting and restarting them as needed. The control system itself is monitored by a subsidiary monitor which corrects and restarts the control system as needed. The architecture employs a persistent database to aid in providing this level of stability, and monitors the functioning of the database system as well. The control system also works to ensure the accuracy of the database records, both by updating them as changes are made and by checking them as needed.

1.2.3 Secure monitoring

The architecture provides for a fairly standard Unix-like scheme of users, groups, passwords, and permissions, specialized for the distinctive classes of operations performed on monitoring networks and their elements.

The architecture also is designed to provide a minimal target for would-be attackers by establishing most of its data communications through ephemeral listeners operating out of randomly-assigned ports. This leaves the only entry points of the system knowable in advance to be the main starting address of the system. Even this can be varied at the time of system startup, and across independent MAITA systems. Future improvements to the system may enable changing of this main address during system operation as well, allowing a form of "frequency hopping".

The initial design of the MAITA system presumes that the most security issues involving the control system must involve mechanisms external to MAITA that provide the operating context of the monitoring processes.

1.2.4 Open monitoring

The architecture is designed to provide an open platform for system development and interconnection. Command operations are transmitted using hypertext transport protocol (HTTP), allowing for basic system operation from any web browser, using commands entered by hand or through multiple specialized web pages or applets. Such web-based control minimizes requirements for installing specialized software on local machines. Supporting this open operation further, we provide reference implementations of the MAITA-specific communications mechanisms, in the form of Java and Common Lisp classes that provide monitoring process wrappers for use in legacy systems written in these languages.

Data are transmitted by an expandable set of common protocols, permitting direct interconnection with many legacy and separately-developed systems. The design permits information to flow through the network by several different protocols, including socket-based ASCII character streams, HTTP (Hypertext Transport Protocol, used by the World Wide Web), SMTP (the Simple Mail Transport Protocol, used by email systems), Java RMI (Java Remote Method Invocation), ODBC (Open Database Connectivity), and OKBC (Open Knowledge Base Connectivity, a protocol for transmitting logical and frame-structured knowledge to and from knowledge bases). the system developer or user chooses the protocol appropriate to the volume, regularity, and type of the information being transmitted. Regular and high-frequency transmissions typically go through persistent stream, ODBC, or OKBC connections. Intermittent and low-frequency transmissions probably go on temporary HTTP, SMTP, Java RMI, ODBC, or OKBC connections. Records of information transmitted to input or from output terminals are structured in protocol-dependent formats.

1.2.5 Intelligent monitoring

One can call any monitoring system knowledge-based, since its designers employ knowledge in the course of its construction. The MAITA architecture is intended to support additional roles for explicitly-represented knowledge, in the operation of monitoring systems.

The first role is that of monitoring processes which explicitly reason in the course of their analysis. MAITA supports these by offering an ontology of monitoring concepts and a knowledge base of monitoring methods. We annotate the structure of information flow with knowledge-level descriptors, distinguishing the *reports* being transmitted and received from the computational representations (*packets*) of these reports, and distinguishing these computational representations from the protocol-specific encodings used for transmission.

The second role is that of monitoring networks, in which the structure of the network explicitly reflects knowledge about the conditions being monitored. This network structure should identify the conditions of interest and the dependencies among them. For example, the structure of a monitoring network should revolve around the conditions on which attention should be focussed, and provide checking of expectations (both positive and negative) related to these conditions.

The third role is that of alerting models, in which knowledge about the likelihood of different classes of alerts or reports, time and other costs of transmission, and utility to different recipients or recipient classes is used to make rational choices about who to tell what, and when and how.

1.2.6 Evolutionary monitoring

Sensible engineering design calls for components that may be reused or adapted in subsequent designs. The MAITA libraries provide means for abstracting, recording, and sharing monitoring networks developed for one purpose with developers of monitors for other purposes. These libraries aim to provide a broad and deep base of abstract and concrete monitoring methods, event descriptions, and alerting models.

Chapter 2

Monitoring concepts

This chapter sketches the fundamental monitoring concepts addressed by the MAITA technology and methodology.

2.1 Information flows

Monitoring systems involve several types of information flows: signals entering monitoring processes as inputs, alerts exiting monitoring processes as outputs, and information that processes send to and retrieve from local stores of memories.

2.1.1 Signals

We think of most information on which the monitoring system operates as type of signals. The types of data sources or signals of interest in monitoring tasks includes continuous signals, sampled continuous signals, text or message streams, propositional information, and graded propositional information, that is, with uncertainty (probability, evidence) or imprecision (fuzzy) measures. The signals of interest vary with the task, and most such tasks will involve only a subset of the possible types of data sources.

2.1.2 Alerts

Alerts represent the immediate results of monitoring, namely the signals sent to humans or other recipients to notify them of the occurrence of some event or the establishment of some condition. The MAITA architecture permits chaining of monitoring processes both sequentially and hierarchically, so that the results of one process can serve as a data source for other processes. Alerts can thus use more abstract languages than the initial input signals, though they may use exactly the same language when acting as pure filters on the input signals.

2.1.3 Local stores and memories

In addition to input signals and output alerts, monitoring processes may also use information in databases or knowledge bases that one interprets as background knowledge or memories instead of signal information. Processes may store statistics or sequences of inputs in memory to perform trend analysis or to facilitate explanations through roll-back and replay. Processes may also change background knowledge in the course of learning more about their environment and task.

2.2 Event recognition

Event recognition proceeds in stages, starting with simple transformations on individual signals to prepare them for more complex correlation and matching operations.

2.2.1 Transducers

Signal transducers transform a signal into one or more new signals. The most familiar variety of signal transducers all concern continuous or time-series signals. These include linear extrapolation and interpolation, trend-line fitting, wavelet decomposition, fourier transforms, summary statistics, outlier detection, threshold detection, and others.

The range of useful signal transducers appears to be more limited for propositional signals, including, for example, translation into new propositions, and measures of change statistics (for example, when last changed, how frequently changing). Sometimes propositions encode data that may be usefully viewed as a sampled signal; for example, sequential reports on the location of a person or piece of equipment may usefully be aggregated into a map-based data series and analyzed with corresponding techniques. One can view many low-level pattern-matching procedures into this category as well, including many internet firewall policies (for example, don't pass any incoming requests from `intruders.net`) and intrusion detection signature-recognizers and statistics collectors (for example, report all write attempts in `/usr/local`, port or IP sweeps, and syslog and ping of death attacks).

2.2.2 Correlators

one can think of signal correlators as multi-signal transducers that take several streams of data as inputs and provide one or more new signals (or propositions) as output. Correlators constitute one of the most important elements of a knowledge-based monitoring system. Events or trends of interest are normally realized in coordinated changes in different aspects of a situation. For example, common statistical abnormality detectors measure discrepancies between statistics at different time scales. As other examples, a discrepancy between rates of use and procurement of a resource can signal a problem needing attention, and determining a new possible motivation for an agent may cast the activity reflected in other data streams in a new light. Plan recognizers also are naturally viewed as looking for specific types of correlations between temporal events that characterize one or more execution paths through the plan. Correlators thus aggregate and abstract input reports to produce more informed output reports.

The building blocks of signal correlators include standard continuous-signal operations such as differencing, modulating, and demodulating, but the most interesting building blocks for knowledge-based applications are those correlating propositional and graded information, such as rules, reasons and argument structures, Bayesian probabilistic networks, causal networks, and temporal constraints.

2.2.3 Trend templates

We use these signal-correlating building blocks in a library of abstract and special signal correlators called *trend templates*, after the representation by that name developed at MIT by Haimowitz and Kohane in the TrenD$_x$ system [7, 6, 13, 10, 4]. A trend template (TT) is an archetypal pattern of data variation in a related collection of data that serves as a characterization of a type of event. For example, a particular information security trend template might characterize an event consisting of a port sweep followed by increased traffic using some particular port to a small set of destinations rarely seen before.

Each TT has a temporal component and a value component. The temporal component includes landmark time points and intervals. Landmark points represent significant events in the lifetime of the monitored process. They may be uncertain in time, and so are represented with time ranges (min max) expressing the minimal and maximal times between them. Intervals represent periods of the process that are significant interpretation. Intervals consist of begin and end points whose times are declared either as offsets of the form (min max) from a landmark point, or offsets of the form (min max) from another interval's begin or end point. The temporal representation is supported by a temporal utility package (TUP) that propagates temporal bound inferences among related points and intervals [12, 11]. The value component characterizes constraints on individual data values and propositions and on computed trends in time-ordered data, and specifies constraints that must hold among different data streams.

2.2.4 Matching methods

In matching a trend template to data, two tasks are carried out simultaneously. First, the bounds on time intervals mentioned in the TT are refined so that the data best fits the TT. For example, a TT that looks for a linear rise in a numeric parameter followed by its holding steady while another parameter decays exponentially must find the (approximate) time boundary between these two conditions. Its best estimate will minimize deviations from the constraints. Second, an overall measure of the quality of fit is computed from the deviations. The measures of quality that tells how well various TTs fit the monitored data become either time-varying signal or propositional outputs of the signal correlators and trend detectors, and provide the appropriately processed inputs for making monitoring decisions.

The most appropriate language of trends and constraints varies from domain to domain. Our original constraint language included mainly linear and quadratic regression models for numeric data, absolute and relative numerical constraints on functions of the data, and logical combinations of such descriptions and propositions. Our newer additions develop the ability to build other TTs using descriptions that characterize any outputs of signal transducers and additional models of correlation among signals. We intend that the template library will grow over time, with research and new applications leading to new additions. Augmenting the library with new templates forms one of the key operations in using the system to address special and temporary concerns.

2.2.5 Recognition control

While one might construct the simplest sorts of monitors to perform operations on input signals in a fixed fashion, event recognition in more complicated situations requires that the recognition operations vary with changing circumstances, that is, that the recognition process possesses a degree of situational awareness and exhibits a degree of situation dependence in its operation. In fact, we can view almost every multi-signal correlation process as exhibiting situational awareness by interpreting some of its inputs in the context of the "situation" represented by the other inputs.

We claim that effective monitoring in many real-world domains requires that at least some of the monitoring processes exhibit situational awareness in a broader sense, in which changes in monitor behavior are triggered by sporadic receipt of updates about a range of relevant conditions occurring in the environment of the monitor. Such updates may take the form of changes to background information rather than receipt of explicitly directed signals. In the information security arena, such updates might be as simple as a change of "infocon" levels akin to "defcon" levels, or as complex as propositional or probabilistic updates to a situational knowledge base maintained by the monitoring process. We can of course view the sequence of such updates as just another input to the monitor, but the sporadic and nonuniform nature and discontinuous effects of such updates, in which they change the way future inputs are processed, make it more natural to view the updates as changing the situational model employed by the monitor in processing the ordinary input signals.

We distinguish several forms of situational awareness useful in monitoring processes. The first dimension of variation divides monitors according to whether the situation in question is an "objective" or"intentional" situation. In the information security domain, the objective situation might consist of information about whether related enclaves are experiencing attacks or whether internet congestion levels seem abnormally high; the intentional situation might consist of the preferences of a security officer regarding the seriousness of abnormalities required to justify issuing an alert on the security desk.

The second dimension of variation divides monitoring methods according to whether the situational model maintained by the process consists of only summary variables or a model of some complexity. In the information security domain, summary variables might include overall probability and disutility of an attack at present, source and target of attack, timing, purpose, and method of attack, and level of defenses available. More complex models might characterize threats using an influence diagram that expresses the overall probability and utility of attack in terms of information about attacks on other enclaves, news articles about increased tensions with known adversaries, and social and infrastructure disruptions such as power outages and strikes.

Degree of passivity forms another important dimension of variation. At one extreme, simple monitors based on lists of indicators and warnings may just observe a set of propositional inputs to detect the presence or absence of a set of specific conditions, and the output of the procedure is to simply report the set of present conditions, or perhaps just the number of conditions present at a given time. At the other extreme, active monitors may start with such a list of indicating conditions and continuously actively seek out new information to determine the presence or absence of these conditions, as opposed to simply waiting for notifications of presence to enter as inputs. Intermediate monitoring procedures might simply filter inputs passively until some threshold is reached, and then switch to an active mode to confirm or deny the remaining conditions.

Degree of passivity is closely tied to notions of the utility of information. Once an active search is underway, the best strategy is to seek first the information most useful to answering the question in the time allowed, but utility considerations also arise in formulating the thresholds at which monitors “go active”. For example, with only a few pieces of information, learning an additional item on an indicators list may not change the quality of the match significantly. But at some point, learning an additional item makes each of the remaining items very significant, and “going active” at that point may well be the appropriate path.

2.3 Event reporting

Useful event reporting relies on informed decisions about what events merit reporting and on perspicuous reporting media.

2.3.1 Alerting control

Recognition of an event calls for deciding what to do with that information: who to notify, when to notify them, and how to notify them. Tailoring the methods used for making alerting decisions constitutes a key method for making the monitoring system responsive to individual analysts. Most of the effort in guiding alerting decisions consists in describing the utility of different results to different agents in different situations.

Alerting decisions depend on the event being reported since analysts have priorities among the conditions of interest to them, and normally wish to hear about the most urgent and important items right away, with the lesser items deferred for consideration later. Alerting decisions depend on the recipient since different analysts will have different interests, priorities, and tasks. Alerting decisions may also depend on the sets of possible recipients and media used to communicate alerts. For example, unreliable transmission or receipt times may call for copying alerts to backup recipients or through alternative transmission media.

2.3.2 Displays

Human analysts require the results of analyses to be presented in intelligible forms, such as graphs and charts that convey the important information prominently without dilution by extraneous detail. The MAITA system presently employs several main display types following common forms, but construction of optimized displays has not been a focus of our effort.

MAITA provides *multivariate strip charts* of selected streams that display the values of one or more variables on a rectangular graph, with the displayed variables plotted vertically and time plotted horizontally. The system can operate individual strips as well as combination strips in which several individual strip charts are stacked one on top of the other with a common temporal reference on the horizontal. The system provides the ability to create combination and multivariate strip charts by selecting various connections or terminals in the process network diagram and then performing the appropriate control-menu operation.

MAITA provides *two-dimensional (2D) maps* of variables plotted against each other that display one or more paired variables, with one set of variables plotted on the vertical, and another set plotted on the horizontal. In a 2D map, time does not appear as a dimension of the graph axes. Instead, the temporal window appears only through the number of points plotted; as the temporal window moves, excessively old points are removed from the display, and the new points are added.

Text alerts display sentences, phrases, or words that constitute alerts to the user.

MAITA also supports combinations of these types. For example, one might combine a strip chart with a text alert that states the normality or abnormality of the stream contents being displayed in the strip chart.

We expect to make future extensions that provide additional visual display types, as well as nonverbal audio and synthesized speech alerts.

2.4 Monitoring knowledge

Different but related bodies of knowledge may be involved in constructing and operating monitoring systems. The monitoring processes themselves may make use of situational knowledge bases for sharing or reasoning about their situations. To aid human developers or analysts in constructing, maintaining, operating, or tailoring monitoring systems, this situational knowledge is less important than knowledge about the varieties of possible monitoring, alerting, and display methods, including descriptions of standard types of events of common interest and means for reporting these events.

2.4.1 Libraries

Libraries of event descriptions, recognition methods, alerting models, and display models constitute the core of the monitoring knowledge of concern to developers constructing new monitoring systems and to analysts tailoring monitoring systems to their their own needs.

2.4.1.1 Event descriptions

There is little to say about the event description library in the abstract since almost everything it contains represents an element of knowledge specific to some domain. The top levels of such event descriptions correspond to the main sorts of events found in everyday language, such as starts, stops, attacks, defenses, infections, cures, weakenings, strengthenings, victories, and defeats. Each body of domain-specific monitoring information adds specific subtypes and instances of such events. Our focus here lies in constructing trend templates to describe interesting and useful event classes, but the library may contain other types of event descriptions as well, such as Bayesian networks that provide probabilistic characterizations of events.

2.4.1.2 Recognition methods

The recognition method library includes entries at all levels of computational detail, from very abstract procedures covering virtually all monitoring tasks, to intermediate-detail procedures capturing more specific algorithmic ideas and domain-specific information about the types of signals being monitored, to highly detailed procedures involving specific representations, code, domain details, and signal sources. For example, the most abstract levels might speak of constructing and comparing a set of hypotheses about what is going on, without providing any details about how the hypotheses are constructed or compared. At an intermediate level, the TrenD$_x$ [7, 6, 13, 10, 4] trend monitoring system developed at MIT uses a partial-match strategy operating over a set of trend templates, each of which consists primarily of temporal constraints characterizing some temporal event. More refined monitoring models would emend this procedure to take probabilistic or default information into account, or to embed background knowledge of the domain in the matching strategy (for example, always try matching location information first before bothering with other information). Still more concrete procedures might describe operating-system-specific methods for recognizing port sweeps or ftp-write attacks. The most abstract control and interpretation procedures serve as a base for more specific ones, but one rarely uses them directly. The real strength of the library of monitoring models lies in identifying specific combinations of representations, procedures, and domain characteristics that offer significant power compared to the more abstract procedures. We expect a fully developed library to contain a great many entries.

Procedures for a small number of fairly abstract monitoring procedures have been codified already in the CommonKADS library of problem solving methods [1], but most of these concern fairly active procedures for diagnosing devices for which complete structural and functional information is available. Such complete information does not exist for some important medical and information security applications.

Combining and cascading monitoring procedures leads to additional library elements, since one may sometimes combine synergistic but separate monitoring procedures into more effective ones. Some of these combinations on monitoring procedures mirror combinations of trend templates, thus reflecting portions of the event description library, but the dataflow connections among monitoring procedures and algorithmic variation in the methods mean that the monitoring procedure library stands on its own rather than as a derivative of the event description library.

Library entries include two types of descriptions of monitoring methods: competence or capability descriptions, and performance descriptions. The capability descriptions characterize primarily the types of information on which the method operates, the relations of inputs to outputs, and the purposes or principal uses of the method. The performance descriptions, in contrast, characterize the representations or data structures used to encode information and the procedural steps or concrete code used to execute the method.

2.4.1.3 Alerting models

The library of alerting models incorporates both extant procedures for making alerting decisions and methods for convenient specification of utility information. The medical informatics literature contains an unsystematic variety of alerting procedures, with few tied to explicit notions of utility (see, for example, [9, 8, 14, 15, 16]). Our ongoing construction of alerting models uses explicit utility models to develop a systematic collection of alerting procedures that includes the ones already reported in the literature. The representations here build on our past work [17, 2, 18, 3] on qualitative representation of utility information, which has developed logical languages that can express generic preferences ("prefer air campaign plans that maintain a center of gravity over those that distribute forces more widely"), and that relate this notion of preference to the notion of problem-solving or planning goals (interpreting goals as conditions preferred to their opposites, other things being equal). We are developing utility models that combine both qualitative preference information with approximate numerical models of common utility structures (for example, utility models that increase up to some time and then drop off to model deadline goals, as in [5]), along with automatic procedures for combining such information into qualitative decision procedures and numerical multiattribute utility functions suitable for quick evaluation of alternatives.

2.4.2 Ontology

If the libraries of event, recognition, and alerting models form the core of systemic monitoring knowledge, the monitoring ontology forms the backbone of both these libraries and knowledge bases that represent the current monitoring situation.

To formalize the monitoring libraries we employ an ontology for monitoring processes, including concepts such as causal structures (chains constitute only the simplest such structures), partial matches, evaluations of significance and likelihood, and focus of attention. This requires a rich language in which to express monitoring and alerting procedures, a broad body of world knowledge with which to relate different monitoring and organizational concepts, and a clear organization of the procedures themselves that reflects abstraction hierarchies and other dimensions along which to classify the procedures. Reference to a body of concepts about the world is essential for specifying the intent of different monitoring procedures, and for increasing the coherence of the library. The organization of the monitoring procedures according to different properties and dimensions of variation is essential for reasoning about and manipulating these descriptions in the course of knowledge acquisition, learning, compilation, and other aspects of the process of maintaining a monitoring system. Similar remarks apply to models of organizational structure and function, which monitors may employ in expressing security policies and judging appropriateness of communications and other operations.

Representing complete domain-specific monitoring situations formally requires even more ontological development, but most of that lies outside the scope of the MAITA project.

2.4.3 Editing tools

To permit easy augmentation and refinement of the set of monitors and bodies of monitoring knowledge and procedures, the MAITA system provides a set of editing tools for creating, copying, removing, filling out, and revising trend templates, monitoring models, and alerting models, as well as a set of informational tools for querying existing ontologies, knowledge bases, and reference materials.

Constructing trend templates involving propositional conditions requires a system for representing these conditions. For practical use, this means having on tap one or more formalized knowledge bases and ontologies to provide the vocabulary and background information needed to express the monitoring conditions. The MAITA system builds on existing tools for editing knowledge bases, augmenting these tools with tailored interfaces for specific types of descriptions, such as event, recognition, and alerting models.

2.5 Monitoring entities

Besides the stuff of monitoring, we identify the means of monitoring in terms of monitoring networks and processes, computational hosts, organizational domains, users and groups, and monitoring system services.

2.5.1 Networks and processes

Monitoring is carried out through the activity of distributed networks of monitoring processes. We permit multiple processes to run on individual hosts, and each such process to involve multiple subthreads. We reify certain subnetworks of processes into named monitoring networks, and treat these as abstract processes, possibly with their own abstract inputs and outputs.

2.5.2 Hosts and domains

We organize the machinery of computation, the host machines and computer networks, into sets we call domains. “Domain” here refers not to the notion of domain name, but rather to a set of hosts defined within the monitoring system. Each domain consists of a set of hosts under the control of or available to some user or users for some purpose, such as all the hosts located within a certain organization, or the subset of those hosts available for monitoring processes. The set of host domains is organized hierarchically using both inclusion and exclusion relations.

2.5.3 Users, groups, and owners

We organize users of monitoring systems along lines familiar from Unix into identified users and groups of users. Users represent individual people, roles, or accounts; groups include zero or more users or subgroups, with the members of a group consisting of its identified members together with the union of the members of its subgroups.

We assign ownership of monitoring resources to individual users and groups, and use ownership, user identity, and group memberships to determine whether a user has permission to perform certain operations.

2.5.4 Services

The MAITA system employs several system-level services in order to support monitoring activities, including directories of libraries of monitoring knowledge and databases of process control and creation information.

Chapter 3

Monitoring architecture

The MAITA architecture was designed to fit easily with existing and new data sources and monitoring systems. To facilitate this, the architecture provides open-ended communication mechanisms for linking with external systems, scalable processing methods, and natural distribution across multiple machines and platforms. Its principal control mechanisms operate via HTTP (hypertext transport protocol), which permits control to exercised through ubiquitous web browsers without the need to install specialized local software.

3.1 Distributed monitoring networks

The central concept in the MAITA architecture is that of a network of monitoring processes under the control of a "monitor of monitors" or "MOM". Such a network consists, at minimum, of the following elements:

- A set, possibly empty, of processes that perform some monitoring task,
- A set, possibly empty, of data communications connections between these monitoring processes, and
- A MOM, which is a particular monitoring process that monitors and controls the sets of monitoring processes and data communication connections.

MAITA monitoring processes communicate information of different types through process-specific input and output terminals, each of which represents a stream of reports of a certain type and meaning. The set of monitoring processes form the nodes of the network, and the communication path between terminals of these processes form the edges or links of the network. The metaphor we use for thinking of the operation of these networks is that of electrical networks, in which we "wire together" various components by their terminals. (Some computer systems refer to similar notions via the publish and subscribe metaphor.) As in electrical networks, the processing network may exhibit a hierarchical structure, as some monitoring processes may consist of a subnetwork of subprocesses.

The notion of terminals is specific to the MAITA architecture, but the architecture permits connection of its own processes with external data sources or recipients as long as the processes constituting those sources or recipients communicate through one of a number of common protocols. In some cases, one may integrate legacy processes more closely into the monitoring network by enclosing them in wrappers to give them terminal functionality of standard MAITA monitoring processes.

Reports flows through the network by different protocols, at present including socket-based ASCII character streams and HTTP, with choice of protocol dictated by the volume, regularity, and type of the information being transmitted. The intent is that regular and high-frequency transmissions go through persistent stream connections, while intermittent and low-frequency transmissions go on temporary HTTP connections. Reports transmitted to input or from output terminals are structured in computational records called packets. Monitoring information packets are unrelated to internet packets, except that the latter are used to transmit the former.

While the initial implementation effort has focussed on stream and HTTP data connections, we expect eventually to support other common protocols, including SHTTP, SMTP, Java RMI, ODBC, and OKBC.

Each process created as part of a MAITA monitoring network is under the control of one or more MOMs, which provide means for constructing, maintaining, inspecting, and modifying the monitoring network and its operation, as described below. As the monitoring process performs certain operations, it notifies each of its MOMs of these actions. We achieve a degree of uniformity in the control process by organizing MOMs as special types of monitoring processes, as befits their function as monitors of monitoring processes.

3.2 Monitoring processes

MAITA monitoring processes include the processes performing the monitoring, the control processes (monitors of monitors, or MOMs) guiding their operation, and display processes for rendering their results to individual users.

Each MAITA monitoring process in the monitoring network possesses

- A control terminal,
- A set (possibly empty) of input terminals,
- A set (possibly empty) of output terminals,
- A set (possibly empty) of subprocesses, and
- A set (possibly empty) of process-network connections.
- A set of at least one MOM possessing authority over the monitoring process.

The function of the monitor process is to respond to commands on its control terminal, and, pursuant to those commands, accept input items on its input terminals and generate output items on its output terminals. The process itself consists of a fluctuating set of computational threads that perform the operations associated with its terminals and connections. The process is controlled and maintained by the MOMs of the process.

3.2.1 Atomic and nonatomic processes

Though processes in general contain the elements just listed, each particular type of process will be either atomic or nonatomic.

Atomic processes are processes without subprocesses. Such processes typically consist of an individual computational process executing on a single host, with subthreads for receiving inputs and handling outputs. Atomic processes necessarily form the foundation of a network of monitoring processes.

Nonatomic processes, in contrast, include distinct atomic subprocesses, either directly, or indirectly through inclusion of nonatomic subprocesses. Such processes typically do not perform any operations on data streams other than to transmit them from their terminals, if any, to the terminals of their subprocesses. Nonatomic processes thus serve mainly to provide conceptual and computational identity to subnetworks of monitoring processes. We do not rule out the possibility of nonatomic processes operating in other ways on their inputs and outputs in addition to communicating them to subprocesses, but do not expect such hybrid processes to be common.

3.2.2 Process states

Each monitoring process may be in a variety of processing states, including ones specific to particular types of monitoring processes. The MAITA system distinguishes a small number of states relevant to controlling monitoring processes.

The *operational state* of a monitoring process is one of `started`, `stopped`, or `paused`. Started processes are operating, and process the inputs they receive. Stopped processes respond to command operations, but do not process any inputs; any inputs sent to them are discarded. Paused processes respond to command operations, and receive inputs sent to them, but do not do anything with the inputs beyond queuing them for later processing.

The *initialization state* of a monitoring process is one of `initialized` or `modified`. Processes start out in the initialized state, and can return to it upon processing an `init` operation (see below). Processes enter the modified state upon processing any command other than an `init` command, and after receiving or processing any input data.

The *functional state* of a monitoring process is one of `functioning` and `failing`. These distinctions refer to the correct functioning of the monitor's operations. Monitors performing their operations correctly are said to be functioning; ones not performing operations or performing them incorrectly or in a partial manner are said to be failing.

The *responsiveness state* of a monitoring process is one of `responsive`, `unresponsive`, and `missing`. These distinctions refer to the monitor's control terminal. Monitors responding to control requests in a normal manner are responsive; ones failing to respond to such requests in a normal manner are unresponsive; and processes entirely absent from the host on which they are supposed to be running are called missing.

3.2.3 Control terminals

Each monitoring process receives instructions through its control terminal. Unlike input and output terminals, the control terminal is an HTTP server with functionality specialized for controlling the monitoring process. Control terminals accept operational commands encoded as a special vocabulary of HTTP requests. The control terminals need not (and typically do not) implement most familiar web server functions (retrieving documents, running CGI scripts, etc.); they need implement only the basic set of monitor-control operations. Each type of process may also implement additional operations particular to its own type. For example, MOMs are themselves monitoring processes handling an additional set of commands in addition to the basic commands common to all monitoring processes.

A control terminal runs as a subthread of its monitoring process, perhaps even as the main thread, listening on its own port for connection requests. Each time it receives a request for a connection, it creates a new subthread to handle the request. The new subthread then accepts the connection and performs the desired operation. Ordinarily these operations will be simple and self-contained, but in some cases they may persist quite some time. This would be the case, for example, with a request to stream out status information as an ever-growing web page ending only when the recipient terminates the connection.

Future extensions to the architecture may remove the limitations of control terminals to make them more like input and output terminals, as described below. In particular, we may someday augment HTTP control communications to permit the use of Java RMI control communications. This extension would make it simpler to issue a command from within the Java MOMs or other programs, and would require the use of an additional port and listener for commands arriving through in Java RMI format. It may also be useful to permit control terminals to be wired to other terminals. Indeed, the current architecture already allows a limited form of this, in which monitoring processes can direct a stream of commands (formatted as HTTP requests) to a control terminal, though this sort of connection treats the control terminal as an "external" system not obeying MAITA architectural conventions.

3.2.4 Data terminals

Each input and output terminal is served by a set of subthreads. Some of these listen for connections on numerous ports, with a separate listener thread and port for each supported protocol (at present ASCII streams and HTTP). Other threads handle individual connections to the terminal, as well as serialization of inputs and servicing of output connections that operate at different speeds.

This can amount to a lot of ports and listeners, depending on how many terminals and protocols are involved. Unfortunately, there does not appear to be any way of making do with a single port or single port per terminal that does not thereby significantly reduce the openness of the architecture. Using a single port (for example, the control terminal) for connections of different protocols would mean having to transmit information across the connection sufficient to identify both the protocol and the terminal to which the connection is to be made. While it is conceivable that one could make a reasonable guess about the general class of protocol being used from the data arriving over the connection, it seems less plausible that one could at the same time transmit information identifying the intended terminal as the initial content transmitted across the terminal. Accordingly, we use different ports to indicate the intended protocol and terminal.

To minimize the number of ports used at particular times, we allow creation of permanent or temporary connection listeners on demand through the `add-protocol` operation. Thus the minimum number of permanent listeners is obtained by only creating temporary listeners just prior to each connection request, and shutting down the listener subsequent to the successful connection. When the monitoring process terminates a terminal listener, it notifies its MOMs with a `protocol-removed` message.

Some apparent redundancies are illusory. In particular, while separate listener ports are redundant for connecting together two monitoring processes designed to have both input and output terminals, they are necessary when connections are desired between monitor processes and non-server systems, such as legacy systems, web browsers, or other external systems.

Each input and output terminal serializes the packets flowing through it. For input terminals, this information may arrive on multiple process-network connections to the terminal. For output terminals, this information may be produced by multiple subthreads of the monitoring process, for example, those operating on information arriving at different input terminals. To effect this serialization, each terminal has a "terminal queue" that acts as a serializer, with all reads and writes to the queue gated by a lock or guard.

3.2.4.1 Input terminals

Each request for connection to an input terminal creates, in a new subthread, a receiver for the connection. The receiver subthread listens for new packets transmitted across the connection, and places each packet of information received on the terminal queue.

3.2.4.2 Output terminals

Each request for a connection to an output terminal places the new connection on a list of connections, with a separate list for each protocol. The terminal queue is served by a subthread that watches for packets appearing on the terminal queue and then transmits the packet across all the connections to the terminal.

Different transmission protocols may have very different time scales. For example, transmission across an existing socket is much quicker than the time needed to insert a new row in a relational database table, and probably much quicker than the time to perform an HTTP transaction. To avoid having slow protocols impede transmission using fast protocols, we create separate threads for communicating packets across each type of connection protocol. That is, we create several new queues, one for each protocol, and have the terminal queue subthread simply place the packet on each of the protocol queues for which there are connections. The subthreads serving each of these protocol queues then transmits the packets across all the connections using that protocol. This scheme may be made more elaborate if there are many connections, in which case it may become necessary to replicate the output queue for each of the slow connections, and to have each slow connection served by its own subthread.

3.2.5 Subprocesses

While many monitoring processes will be constituted as ordinary computer processes performing actual computation, we also allow monitoring processes to consist of networks of subprocesses. Descriptions of such network fragments provide a convenient way of storing useful patterns of processing, and reduce the effort needed in constructing a desired network of processes. These subprocesses are full-fledged monitoring processes, responding to all the basic monitoring process commands.

A monitoring process that has subprocesses gives each subprocess a name or identifier that allows unique reference to the subprocess with respect to the parent process. This name or identifier need not be unique in any other context. This uniqueness restriction permits one to refer to a subprocess of any depth by a pathname, for example, "the A subprocess ... of the Z subprocess".

3.2.6 Subprocess connections

Processes implemented as networks of subprocesses must specify the data connections between these subprocesses. We view such process-network connections as internal components of the process, in distinction to other connections linking terminals of the process or of its subprocesses with terminals outside of the internal subnetwork.

A monitoring process that has subprocesses need not have any input or output terminals of its own. In such a case, the process provides a name for the network fragment comprised by its subprocesses and network connections. One then uses the process by making connections to the input and output terminals of the subprocesses. However, a monitoring process with subprocesses may also have its own input and output terminals. These can be used or viewed as providing a level of abstraction over the subprocess network structure. In this usage, the subnetwork includes connections from the parent process inputs to input terminals of the subprocesses, and from output terminals of the subprocesses to the parent process outputs. The MAITA architecture at present provides no automatic mechanisms for hiding such subnetworks; all subprocesses at all levels are open to connection at the instruction of the user. The permissions system described below may be used to make some or all processes owned by one user invisible to other users.

Future revisions of the architecture may incorporate mechanisms for controlling the visibility and accessibility of subnetwork structures, along the lines of familiar mechanisms in standard programming languages.

3.2.7 Port assignments

Since a given host machine may be called upon to run several instances of the same type of monitoring process, and since different hosts may already have allocated different sets of ports to processes running on them, monitoring processes cannot rely on any scheme of fixed port number assignments for control and data terminal listeners. Accordingly, we dynamically allocate port numbers on each host at the time of process creation, and communicate these assignments to the monitoring system in order that the MOM and other processes may know where the process is listening. This operation is called registration, and is effected with the MOM command `started`.

Unix provides means for identifying unused ports as a basic system operation. For operating systems that do not provide such a service, both Lisp and Java provide means for trapping port-assignment errors, so one may write code which locates an unused port and sets up a listener on it. One caution here is that using programming-language routines to search for unused hosts may well trigger security mechanisms looking for port-sweep intrusions.

3.3 Inter-process connections

We wish to speak both of internet connections, via sockets, as well as links in the monitoring-process wiring diagram. We call the former connections, and the latter process-network connections. Implementation of a process-network connection may involve a single internet connection, in the case of persistent ASCII streams, or multiple internet connections, in the case of HTTP transmissions of successive packets.

Process-network connections are specified by indicating two terminals to be wired together. It is an error to connect two terminals with different associated packet types. One may wire together two input terminals I1 and I2, respectively of processes P1 and P2, only when P2 is a subprocess (recursively) of P1. Similarly, one may wire together two output terminals O1 and O2 of processes P1 and P2 only when P1 is a subprocess (recursively) of P2.

Note that feedback in the process network is permissible as long as this makes sense semantically given the processing done by the monitors involved. As an example, consider a monitor in which the initial filter uses a threshold that depends on the perceived criticality of the health of the patient (that is, how sick he is), and receives updates to this criticality value on some input terminal. It is entirely reasonable that some downstream monitor send out alerts about the overall criticality of the patient, which are fed back into the initial transducer to update the filter threshold.

Note that mere monitoring network data connections does not actually initiate any processing of data streams. The user must also issue (or ask the MOM to issue) `start` commands to the processes so connected to initiate processing of the data streams.

3.3.1 Data communication packets

Primary data transmission among monitoring processes occurs by using collections of information called *packets*. Packets are essentially multi-level record structures. We distinguish packets from their encodings for transmission by different transmission protocols. We also distinguish packets from their meanings to the sending and receiving processes. At the knowledge level we think of monitoring processes as trafficking in *reports*, which we describe in terms of knowledge-level concepts. Packets merely represent encodings of reports in terms of standard computational data types.

The same packet of information may be transmitted to different receiving processes by any of the several transmission protocols. Though each of these protocols requires different encodings of the packet information, the transmission conveys the same information no matter what be the protocol.

The same packet of information may mean different things to different processes. For example, one process may interpret packets containing only patient names and identifiers as reports identifying patients who have just been born, while another process may interpret information arriving in exactly the same packet structures as reports identifying patients who have just died. Another process may interpret packets containing an O_2 level as reports of the current O_2 level of a patient, while the same type of packet may indicate reports of an average O_2 level over the past hour, or even a target O_2 level for ventilation control, when received by a different process, or even by a different terminal of the same process. The information source and target terminals determine the packet interpretations, not the packet record structure or constituent data types.

The basic types of information that may be included in packets are numbers, booleans, characters, strings. In addition, packets items may consist of lists of such values, or lists of lists, etc. Numbers are restricted to types and values representable in Java, that is, short, medium, and long integers, and single and double-precision floating point numbers. String values can be of arbitrary length, and may contain any character excluding the ASCII null character.

3.3.2 Data communication protocols

The MAITA architecture supports a variety of different data communication protocols. The architecture does not limit the set of protocols that might be supported, but the current implementation only provides support for a limited set at present, namely ASCII character streams and HTTP (hypertext transport protocol). We expect the implementation to extend to include SMTP (simple mail transport protocol), Java RMI (remote method invocation), ODBC (open database connectivity), and OKBC (open knowledge base connectivity).

Each protocol caters to different assumptions about data rates and implementation languages. High volume data streams benefit from persistent connections; to this end, the first protocol communicates packets as ASCII character strings sent along a stream socket connection. Intermittent or low-volume transmissions may be better sent via HTTP requests, or as Java RMI requests for processes implemented in Java. Alerts to humans normally go via SMTP, if not displayed by graphical means. Database and knowledge-base connections use ODBC or OKBC protocols.

3.3.3 Packet information encodings

Each data communication protocol requires different encodings of packet information. We do not specify the encodings of Java RMI, ODBC, or OKBC information here, as those are specified at length elsewhere.

3.3.3.1 ASCII character streams

Encodings of a packet in an ASCII character stream consist of a sequence of ASCII characters terminated by a null character (ASCII 000). The encoding character sequence consists of encodings of each field of the packets separated by whitespace (spaces, tabs, linefeeds, carriage returns). Arbitrary amounts of whitespace may precede or follow any field, including the first and the last. Normally, there will be no whitespace preceding the first or succeeding the last, and only a single space separating adjacent field encodings.

We permit packets to express silence on a field of any type by using a lone question mark (`?`) to represent the unspecified or missing value of that type.

Numerical fields are encoded either directly in decimal numerical representations, or with exponential expression, for example, `3.14E-218`. Leading plus and minus signs are permitted, as are leading zeros (on both the mantissa and exponent in exponential notation).

The lone characters `T` and `F` encode true and false boolean values.

With one exception, values of character type are single ASCII characters enclosed by a pair of double quotes (`"`). The double quote character itself is included, encoded by the three character sequence `"""`. The exception is the ASCII null character, which is encoded as the two character sequence `""`.

String values are enclosed by a pair of double quotes. The sequence `""` represents the empty string. When appearing in strings, double quotes and backslash (`\`) must be preceded by backslash, which serves as an escape character. Linefeeds and tabs may appear in strings. Null characters are not allowed in strings.

List values are encoded using parentheses to delimit the list, as in `(v1 v2 ... vn)`, with the parentheses enclosing the list element encodings. List elements are separated by whitespace, following the same rules as field encodings stated above.

3.3.3.2 HTTP

Packets transmitted by HTTP will be transmitted as either GET or POST methods, depending on the amount of information conveyed by packet type. GET methods can transmit small packets reliably, but packets with long encodings require POST methods for reliable transmission. If a uniform method is needed, POST methods should be used.

The encoding of packet element values for HTTP transmission is obtained by starting with the text stream format and then using standard MIME encoding of the special characters. One then uses standard HTTP syntax for expressing the request in terms of these values. Neither the commands nor the argument names are case sensitive in the MAITA system.

3.4 Process control protocol

Monitoring process control operates by transmission of control requests to the control terminals of monitoring processes, followed by transmission of control responses back to the source of the control request. As mentioned earlier, the MAITA system employs HTTP for both directions of this control protocol. Mere use of HTTP implies no special similarity to or reliance on standard web-server operations. Monitoring processes do not typically retrieve files, process forms, etc. We use HTTP purely as a protocol for transmitting requests and responses to these requests. The MAITA control protocol currently operates using the HTTP protocol version 1.1.

We use HTTP in this way both because it provides an established protocol for back-and-forth information exchange, but also to increase the openness of the system. HTTP allows commands to be issued either manually by a person with a web browser, or automatically, by web pages, applets, or other processes. This permits one to construct multiple interfaces to the underlying system of different complexity. Because it is inconvenient to manually issue commands through web browsers, the architecture requires that each process offer a control panel accessible via the root HTTP command. We do not specify just what operations this default control panel should provide, and do not require all processes to provide the same functionality in their control panels. Typical control panels might provide indications of current processing parameter values and means for changing these parameter values. Common processing parameter types might include thresholds, scale factors, and tolerances.

3.4.1 Control requests

Each HTTP control request is answered by an HTTP control response. Each monitoring process must handle the standard vocabulary of monitor operations detailed below in Section 3.6, and may also respond to additional types of requests, at the option of the designer of the monitoring process.

Command operations are transmitted as HTTP GET or POST commands. As in the case of data packet transmission, short operation requests may safely be sent via GET commands, while longer requests should be sent via POST commands.

Each operation descriptor starts with the prefix `/maita`, which we omit below for the sake of brevity. This prefix isolates the MAITA commands from the usual vocabulary of web servers. It may someday permit incorporation of monitoring functionality into standard web servers without disrupting standard functionality. Control requests are *not* CGI requests; for illustration,

```
http://maita.lcs.mit.edu:1492/maita/start
```

would issue a `start` request to the process listening on port 1492 on the host maita.lcs.mit.edu.

3.4.2 Control responses

HTTP specifies that the response to a request is transmitted over the same network connection that transmitted the request itself. Because some responses may involve streaming out data over an indefinite period, response connections may be quite long-lived. For example, a command may request a list of results, to be incrementally extended as processing continues. The command operation server must support such long-lived responses, though in this illustration it may make more sense to organize the monitoring process to emit the list of results on an output terminal.

The control operation server of a monitoring process does not perform the usual operations provided by ordinary web servers, only those functions that are necessary for the operation of the monitoring system. In particular, these processes give "`Forbidden`" error returns when asked for arbitrary documents. Though some monitors may make particular files available (for example, help files or instruction pages), they generally won't provide access to the file system.

All requests other than one of the legal requests listed below or particular to the specific monitor receiving the request result in one of the responses

```
400 Bad Request
403 Forbidden
```

```
404 Not Found
501 Not Implemented
```

The `Bad Request` response comes back if the request has some syntactic error. The `Forbidden` response is returned for operations not allowed by the monitoring process, which ordinarily includes virtually all of the ordinary requests one sends to web servers, such as `/cgi-bin/` requests, file retrieval requests, etc. The `Not Found` response is appropriate for requests for documents which prove missing. The `Not Implemented` response is given for some request that is syntactically acceptable and in principle legal, but for which the implementation does not provide mechanisms.

The responses to successful performance of a legal operation request may vary with the type of operation or response, and should be one of the responses

```
200 OK
205 Reset Content
204 No Content
```

The `OK` response is appropriate for fixed responses. The `Reset Content` response is appropriate for responses that may differ each time the operation is performed. The `No Content` response is appropriate for operations which do not return any performance information. While `No Content` may be the natural choice in most cases, we have found it useful to provide summary information about the performance of the operation in the response, as an aid to debugging and system administration. Response information is especially helpful in making it possible to use a web browser to issue monitor-control or inspection commands.

If the performance of some legal operation fails for some reason, then the response should be a 400 or 500 series HTTP response, chosen appropriately. The usual candidates are

```
409 Conflict
500 Internal Server Error
503 Service Unavailable
```

The `Conflict` response indicates some conflict between the requested operation and some aspect of the state of the process. The response signals an `Internal Server Error` when something goes wrong in processing the operation. The `Service Unavailable` response is appropriate for operations which are only temporarily unsuccessful.

3.4.3 Request types

Each monitoring process is required to respond to a set of basic command operation types, which we divide into five categories:

- *Informative* operations, which provide information about a process, its role in the network, and its operating status;
- *Read* and *write* operations, which initiate or modify data connections between monitoring processes;
- *Operational status* operations, which start, stop, or otherwise control processing performed by monitoring processes; and
- *MOM* operations, which govern relations between the process and MOMs.

3.4.4 Permissions

Our initial implementation strives for openness, at the expense of security. In particular, any MOM can change any aspect of the system organization visible to it. This makes it possible for different users to destructively interfere with each other's activities.

To minimize interference between users and to increase the security of the system, we associate permissions for different classes of operations with different users. The scheme here is modeled on that used by UNIX for file system privileges.

Rather than assign permissions for individual monitor operations, we group the standard operations into classes, and assign permissions based on the class of the operation. We use the letters `irwom` to refer to these classes of operations, much as Unix uses `rwx` to refer to its three classes of file system operations. Specifically, the classes of informative (`i`), operational status (`o`), and MOM (`m`) operations are as given above. The class of read (`r`) operations consists of connecting to or disconnecting from an output terminal; the class of write (`w`) operations consists of connecting to or disconnecting from an input terminal.

Each user and group may be accorded particular permissions by a process, both for the process itself, and for the process terminals. That is, the informative, operational, and MOM permissions apply to the monitoring process as a whole. The read and write permissions apply to individual terminals, so that, for example, a monitoring process may let anyone read from one terminal, but only designated recipients read from another.

Accordingly, a permissions specification for a monitoring process may be represented by a string of fifteen such letters and hyphens, with hyphens indicating lack of permission for a class of operations, and with a group of permission abbreviators given for each class of user: five letters for the owner, five for the members of the group, and five for other users. For example, a permissions string representing full permissions to the owner and the owner group but no others would be

```
irwomirwom-----
```

Every monitor control operation may be accompanied by `login` and `password` arguments to provide authentication of the source. By default, omitting these arguments when issuing a command corresponds to “anonymous” login; different processes may handle such requests differently. We assume that communications between a process and one of its MOMs do not require authentication, as the origin host and port of the communication provides the needed authentication. Monitoring processes receiving requests for connections from others may authenticate the request by sending a `verify-permission` query to a MOM, passing along the required permissions together with the requester’s login identifier and password and receiving verification of the requester’s identify and group membership.

3.5 Monitoring process creation

Monitoring processes may be created either manually by means external to the MAITA system, or by action of a MOM. The former case may include setup of legacy systems operating through MAITA process wrappers. The latter case is the normal case for processes drawn from the MAITA monitoring library, though even these may be created manually as well.

In either the manual or MOM method of process creation, the MOM must be notified of the location of the new process’s control terminal through a `created` notification. The wrapper used in processes created from the monitoring library automatically performs such a notification when such processes are created. For processes created manually prior to the existence of a MOM, the user desiring to include the process in the monitoring network must send this notification to the appropriate MOM.

3.5.1 Process creation servers

Creation of a monitoring process on a host currently requires that the process be defined as an executable on that host. We use the term “script” to mean the executable mechanism for process creation whether the item is a binary executable, a Perl script, or some other web-server request mechanism. Ideally, the code would not require local installation, but could be downloaded over the network as needed. Doing this requires security mechanisms not yet available to us.

Lacking reliable means at present for remotely executing an application on a host, we require that a process-creation server be running on each host on which monitoring processes are to be run.

We require nothing of this server except that it accept HTTP requests and handle a specific set of requests in CGI format, namely requests to create processes and requests to determine whether the server is operating properly or not. For convenience we use a standard web server (Apache) as the process-creation server. We define a CGI script that takes the process-creation arguments listed below and sets a monitoring process in motion. We define other CGI scripts to handle the server status-checking operations.

A full-blown web server involves far more complexity than is required for process creation, especially in light of the amount of system administration typically needed to install, configure, and maintain a standard web server. Future implementations might well employ a specialized server application that directly performs only the necessary process-creation and status-checking operations and rejects all other requests as unimplemented. No matter which approach is taken, ensuring that the server is always running on the host machine requires a significant system administration effort.

There is no requirement that the server listen on the same port no matter what the host. To avoid the need for such uniformity, the MOM records locations of listeners and scripts on each host.

3.5.2 Process script locations

The set of installed scripts may be located anywhere on the host as long as the location is accessible to the server, since the MOM script table provides the mapping between the process type name and the location of the script on a given host. In particular, the scripts need not be located all in the same place, but may be scattered throughout the file system of the process-creation server. Ideally, the set of installed scripts is managed using a version control system, but we do not present such here.

For convenience, we store all scripts in a single directory. When the file system permits, we use the names of the process types from the process library to name the files containing the corresponding scripts. When the file system does not permit use of any legal process name as a filename, we generate filenames of the form `<ll>-nnnnn`, where `<ll>` is the first two letters of the name of the monitoring process type, and `nnnnn` is a string of five digits. To facilitate manual inspection of the set of creation scripts, we also create a file with the name `<ll>-nnnnn.txt` in the same directory as the script and store the name of the monitoring process type in that file.

3.5.3 Process creation arguments

Whether process creation is handled by a full web server or a server specialized for this purpose alone, we communicate creation requests to the server using HTTP, as in the request

```
/cgi-bin/maita/ma-00001?login= ...
```

Creation requests employ the following arguments to specify the process.

3.5.3.1 Required creation arguments

login=`<uname>`
This identifies the source of the request.

password=`<string>`
This authenticates the source of the request.

script=`<string>`
This provides a pathname in the local file system of an executable script or application which, when executed, sets up the desired monitoring process.

host=`<hostid>`
This indicates the host on which the MOM creating the process is running, so that the new process can acknowledge its creation to the MOM.

port=`<portid>`
This indicates the port on which the MOM creating the process is listening, so that the new process can acknowledge its creation to the MOM.

base=`<base>`
This provides the base URL component for the control listener of the MOM creating the process. It is required only when there is a nonempty base used in the MOM control terminal URL.

owner=`<uname>`
This identifies the owner of the process for interpreting permissions.

group=`<uname>`
This identifies the group of the process for interpreting permissions.

permissions=`<permstring>`
This identifies the permission string for the process.

3.5.3.2 Optional creation arguments

ohost=`<hostid>`
This indicates the grandmom host.

oport=`<portid>`
This indicates the grandmom port.

obase=`<base>`
This provides the grandmom base URL component.

id=`<id>`
The `id` is the process ID assigned by the MOM to the new process. This value may be supplied later via the `setid` monitor command.

reqport=`<portid>`
The value of `reqport` is a required port number for the process control terminal. Requiring a specific port is generally not a good idea, as it may prevent creation of the process when multiple instances of the same process are set to run on the same machine.

3.6 General monitor control operations

Each monitoring process must accept and perform the basic set of requests listed below, though the work required in performing some of these operations may vary among monitoring processes. Some of these operations will notify a MOM of changes effected in the course of performing the operation, using MOM commands described elsewhere.

3.6.1 Operation arguments

Each of the standard monitoring process control operations accepts a subset of the following types of standard arguments:

- Operation scope arguments: `scope`
- Connection type arguments: `source`, `target`, `iotype`, `ctype`
- Information access arguments: `host`, `port`, `base`, `ohost`, `oport`, `obase`, `protocol`, `version`, `login`, `password`, `ologin`, `opassword`, `email`, `serverstring`, `nativestring`
- Operation response arguments: `response`, `html`

- Operation permission arguments: `owner`, `group`, `permissions`
- Miscellaneous arguments: `time`, `id`

We explain each of these standard arguments separately, along with its usual interpretation or role in standard monitoring process control operations, using HTTP parameter-passing syntax.

Commands issued to monitoring processes may also contain other arguments, but all processes are free to ignore any arguments beyond the ones their required or optional arguments.

3.6.1.1 Operation scope arguments

scope=`<scope>`

The `scope` specification is used to indicate the scope of operations across subprocesses of the process receiving the operation. The syntax is given by

```
<scope>       ::= all | self | subs | <procid>
<procid>      ::= <name>  | <pathname>
<name>        ::= --string of letters, digits, underscores,
                        and hyphens, but not periods--
<pathname>    ::= <name>  | <name>.<pathname>
```

In such specifications, "`self`" means apply the operation only to the receiving process itself, and "`subs`" means apply the operation only to the subprocesses (recursively) of the receiving process. A `procid` designates some subprocess of the process receiving the operation. The possible designators are an integral ID number assigned by the process, the symbolic name for the subprocess used by the process, or a pathname composed of these two sorts of designators.

```
scope=O2-detector
scope=2
scope=A.B.2.O2-detector
```

Here the pathname `A.B.2.O2-detector` denotes the the `A` subprocess of the `B` subprocess of the 2 subprocess of the `O2-detector` subprocess. We allow arbitrary amounts of intra-line whitespace (spaces and tabs) to separate the pathname elements.

duration=`<duration>`

The `duration` specification is used to indicate the temporal extent of objects. The syntax is given by

```
<duration>    ::= once | indefinite | <interval>  | <until>
<interval>    ::= (<int-units> <number> )
<int-units>   ::= seconds | minutes | hours | days
<until>       ::= (until <time> ) | (until successful)
```

In such specifications, "`once`" means the object should exist through one operation and then terminate. "`Indefinite`" means the object should exist until explicitly terminated. "`Interval`" durations specify existence for a temporal interval with the specified length. "`Until`" specifications specify existence until the indicated time, or until successful completion of an operation.

3.6.1.2 Connection type arguments

source=`<termid>`

A source argument indicates the source terminal or location in specifications of connections among terminals or between terminals and external locations.

The `termid` syntax

```
<termid>      ::= <name>  | <pathname>
```

mirrors that of `procid`, but the interpretation is somewhat different. Here the simple names designate terminals of the process, while pathnames are interpreted so that the first item of the pathname denotes a terminal, while the subsequent items of the pathname denote subprocesses of the process receiving the command. As a terminal identifier, the pathname `A.B.2.O2-detector` denotes the the `A` terminal of the `B` subprocess of the `2` subprocess of the `O2-detector` subprocess.

target=`<termid>`

A target argument indicates the destination terminal or location in specifications of connections, with an interpretation parallel to that of source arguments.

iotype=`<iotype>`

This argument specifies the communication direction of terminals, with a syntax:

```
<iotype>      ::= i  | I  | input  | o  | O  | output
```

ctype=`<ctype>`

This argument specifies a type of connection, with the syntax

```
<ctype>       ::= ii | io | oo | ei | oe | internal | external
```

The `ctype` values "`ii`", "`oi`", or "`oo`" denote connections among terminals of processes: respectively an input-to-input, output-to-input, or output-to-output connection. The values "`ei`" and "`oe`" indicate connections to external systems. Such systems are not assumed to have terminals (though they may), and require a different means for specifying them. The value "`ei`" denotes a connection from an external data source to an input terminal; "`oe`" denotes a connection from an output terminal to an external target. The value "`internal`" means the union of the "`ii`", "`oi`", and "`oo`" types; the value "`external`" means the union of the "`ei`" and "`oe`" types.

3.6.1.3 Information access arguments

host=`<hostid>`

This argument specifies a host machine or address on the internet, using the syntax

```
<hostid>      ::= <ipaddress> | <domain name>
```

A `hostid` is either a dotted IP address (for example, 18.30.0.179) or a symbolic IP name (for example, lcs.mit.edu).

ohost=`<hostid>`

This argument specifies a second host in commands that need to specify two hosts.

port=`<portid>`

This argument specifies a port number for communication protocols, using the syntax

```
<portid>      ::= <integer>
```

A `portid` is an integer between 0 and 65535, inclusive.

oport=`<portid>`
This argument specifies a second port number.

base=`<base>`
A `base` argument specifies a base URL component to use as a prefix for connection requests, using the syntax

```
<base>        ::= <string>
```

If a host and port are also specified, the connection request goes to the URL `<host>:<port>/<base>`.

obase=`<base>`
This argument specifies a second base URL component.

protocol=`<protocol>`
The `protocol` argument designates one of the types of communication protocols used for data communications, using the syntax

```
<protocol>    ::= stream | ssl | http | shttp |
                  smtp | jrmi | odbc | okbc
```

“`Stream`” means socket stream connections; “`jrmi`” means Java RMI.

version=`<version>`
Version arguments specify versions of the protocols used, with the syntax:

```
<version>     ::= <string>
```

The exact nature of `version` strings is not spelled out here. The idea is that these are alphanumeric strings with period (`.`) separators, such as “`1.1`”, as would be used in specifying the protocol HTTP 1.1.

login=`<uname>`
This argument supplies a user name for operations requiring login dialogues.

```
<uname>       ::= --string of letters, digits, underscores,
                     hyphens, percent signs, and periods--
```

ologin=`<uname>`
This argument supplies a second login identifier for commands that require specification of two users.

password=`<string>`
This argument supplies a password for operations requiring login dialogues.

opassword=`<uname>`
This argument supplies a second password value for commands that require specification of two users.

email=`<email>`
The `email` argument provides an email address for data sent via SMTP, using the usual syntax:

```
<email>       ::= <uname>@<hostid>
```

User names may include any of the characters standardly allowed for such names.

serverstring=`<server connect string>`

A `serverstring` argument indicates a string used to make connections to specialized servers, for example, a ODBC connect string. We do not specify any syntax here.

nativestring=`<native connect string>`

A `nativestring` argument indicates a string used to make connections to specialized servers, for example, a native DB connect string. We do not specify any syntax here.

3.6.1.4 Operation response arguments

response=`<response>`

This argument specifies whether a response is desired, and if so, of what length, using the syntax:

```
<response>    ::= <yes-or-no>  | short | medium | long
<yes-or-no>   ::= yes | no | y | n
```

The `response` argument is "`yes`", the operation should provide a response of some form. If the argument is "`no`", or if the variable and value are not specified, the operation may return a `No Content` response. If the argument is "`short`", "`medium`", or "`long`", then a response should be returned (as with "`yes`"), with a level of detail as suggested by the argument name.

html=`<yes-or-no>`

This argument specifies whether HTML formatting of any response is desired. If the `html` argument is `yes`, then the response should be formatted as an HTML document for viewing by a browser. If the argument is `no`, or if the variable and value are not specified, any response may be formatted as plain text.

3.6.1.5 Operation permission arguments

owner=`<uname>`

This argument specifies the identifier of the owner of a process.

group=`<uname>`

This argument specifies the identifier of the group of owners of a process.

permissions=`<permstring>`

This argument specifies the permissions granted to the different users and groups regarding different classes of operations. process.

```
<permstring>  ::= <owner-perms> <group-perms> <other-perms>
<owner-perms> ::= <perms>
<group-perms> ::= <perms>
<other-perms> ::= <perms>
<perms>       ::= <iperm> <rperm> <wperm> <operm> <mperm>
<iperm>       ::= i | -
<rperm>       ::= r | -
<wperm>       ::= w | -
<operm>       ::= o | -
<mperm>       ::= m | -
```

The `ownerperms`, `groupperms`, and `otherperms` indicate respectively the permissions granted to the owner, to the owner's group, and to others. Each permissions specification grants permission for a specific class of operations by use of `i`, `r`, `w`, `o`, or `m`. Permission is denied for a class of operations by giving the value `-`.

3.6.1.6 Miscellaneous arguments

time=`<timestamp>`

This argument supplies a time value along with a request, using the syntax:

```
<timestamp>   ::= <string>
```

A `timestamp` is a string of the form HH:MM:SS:DD:MM:YYYY:ZZ.Z:D The second MM is month number, the ZZ.Z is the time zone (some are half-integer) or the string "`UTC`", D is "`S`" or "`D`" depending on whether the time includes Daylight Savings Time. If ZZ.Z is "`UTC`" then the final D specification may be omitted. Days, Months, and Years are all 1-based, not 0-based.

id=`<id>`

This argument provides an integer used by a MOM to identify a process, terminal, or connection, using the syntax:

```
<id>          ::= <integer>
```

3.6.2 Operation types and syntax

Each monitoring process is required to respond to a set of basic command operation types, which we divide into four categories:

- Informative operations: `about`, `help`, `terminals`, `ports`, `subprocesses`, `connections`, `owner`, `group`, `permissions`, `moms`, `grandmoms`, `status`, `ping`
- Read/write operations: `add-protocol`, `remove-protocol`, `connect`, `disconnect`
- Operational status operations: `controls`, `/`, `start`, `stop`, `pause`, `resume`, `init`, `quit`, `checkpoint`
- MOM operations: `setid`, `setowner`, `setgroup`, `setpermissions`, `add-mom`, `remove-mom`, `replace-mom`, `add-grandmom`, `remove-grandmom`

We explain each of these operation types and its syntax in the list below.

Each command takes various arguments, some required, some optional. We indicate the legal arguments in a list after the command name, enclosing the optional arguments in square braces, as in

```
/operation r-arg1 ... r-argM [o-arg1 ... o-argN]
```

Here the `r-arg` items denote the required arguments, while the `o-arg` items denote the optional arguments. The order of arguments does not matter since HTTP uses keywords for parameter passing.

Beyond the arguments listed explicitly below, optional arguments `login`, `password`, `html`, `response`, and `time` are always allowed and sometimes required, as noted below.

3.6.2.1 Informative operations

/about

This returns a description of the process from the library and process documentation.

/help

Returns a page giving instructions on how to use the monitoring process. If the process is a MOM, this page provides information on how to use the system, together with pointers to general MAITA system information.

/terminals [**iotype**]

Return a table of terminal information, in the following format:

```
<n> terminals
<id-1> <name-1> <I/O-1> <host-1> <base-1> <pkt-1> <perms-1>
...
<id-n> <name-n> <I/O-n> <host-n> <port-n> <base-n> <pkt-n> <perms-n>
```

Here `id`'s are identifying integers assigned by the process; `name`'s are symbolic name strings; `I/O` is one of `I` or `O`, indicating input or output; `host`, `port`, and `base` specify a URL for the terminal listener. The `pkt` entries give the names of the packet types used by the terminals. The optional `iotype` argument indicates whether to return only `I` or `O` terminals instead of the full list. The `perms` give a permission string indicating the operations permitted on the terminal.

/ports [**terminal iotype protocol version**]
Return a table of terminal port assignments, in the following format:

```
<id-1> <I/O-1> <port-1> <protocol-1> <version-1>
...
<id-n> <I/O-n> <port-n> <protocol-n> <version-n>
```

Here the `id`'s are the identifying integers assigned by the process; `I/O` indicate whether an input or output terminal; and the `port`, `protocol`, `version` entries indicate the ports used for connections according to the different protocols and versions. The optional arguments, when supplied, indicate to return only the table entries for the stipulated terminal, iotype, protocol, or version.

/subprocesses
Return a table of direct subprocesses (but not subsubprocesses etc.), in the following format:

```
<n> subprocesses
<id-1> <type-1> <host-1> <port-1> <base-1>
...
<id-n> <type-n> <host-n> <port-n> <base-n>
```

/connections [**ctype**]
Return a table of internal process-network connections, in the following format:

```
<n> connections
<source-1> <target-1> <ctype-1> <protocol-1> <version-1>
...
<source-n> <target-n> <ctype-n> <protocol-n> <version-n>
```

Here the `source` and `target` values are pathnames in the case of internal terminals, and are URLs in the case of external locations. The `ctype` values are one of the values listed earlier (`ii`, `oi`, `oo`, `ei`, `oe`, `internal`, `external`). If one of these values is given as an argument, the table returned includes just those connections of the specified type.

/owner
Returns the name of the owner of the process.

/group
Returns the name of the owner group of the process.

/permissions

Returns the permissions table for different operations associated with the process.

```
Owner: <owner-perms>
Group: <group-perms>
Other: <other-perms>
```

Information about the permissions associated with any terminals of the process may be obtained via the `terminals` command.

/moms

This returns a table of the MOMs for the process in the following format:

```
<n> MOMs
<host-1> <port-1> <base-1> <name-1>
...
<host-n> <port-n> <base-n> <name-n>
```

/grandmoms

This returns a table of the grandMOMs for the process's MOMs in the following format:

```
<n> grandMOMs
<mhost-1> <mport-1> <mbase-1> <ohost-1> <oport-1> <obase-1>
...
<mhost-n> <mport-n> <mbase-n> <ohost-n> <oport-n> <obase-n>
```

Here the `m` entries identify a MOM, while the corresponding `o` entries identify the grandMOM for that MOM.

/status [**response**]

Provide information about the operating setup and status of the monitor. Not all levels of detail need be distinct. If the detail level is omitted, `response=short` is assumed.

The `short` response detail level returns a table of information of the form

```
Process <name>
Type <ptype>
Status <operational-state>
Functionality <functional-state>
Created <timestamp>
Host <host>
Port <port>
Base <base>
Protocol <protocol>
Version <version>
GrandmomHost <host>
GrandmomPort <port>
GrandmomBase <base>
```

The status report here will be one of `started`, `stopped`, or `paused`. The functionality report here will be either `functioning` or `failing`. The process may augment this information with process-specific status information, appearing after the information above is presented.

The `medium` response detail level provides the information given by the `short` level, and appends to this information the information given by the `terminals`, `ports`, `subprocesses`, `connections`, `moms`, and `grandmoms` commands. The process may augment this information with process-specific status information.

The `long` response detail level provides the information given by the `medium` level, and appends to this the information given by the `about` command. The process may augment this information with process-specific status information.

/ping
A synonym for `/status?response=short`, intended to provide confirmation that a process is operating.

/report-parameter name
This requests the process to provide the value for the operational parameter named by `name`.

3.6.2.2 Read/write operations

/add-protocol termid protocol [version duration login password]
Add a listener to a terminal that listens for connections using the specified protocol (and optionally, version). The `duration` argument indicates how long the listener should operate. By default, listeners operate indefinitely. The optional `login` and `password` arguments can be given to restrict connections to those using the correct values. Notifies its MOMs with a `protocol-added` notification.

/remove-protocol termid protocol [version login password]
Removes the listener from a terminal listening for connections using the specified protocol (and optionally, version), if any, using the `login` and `password` arguments, if supplied. Notifies its MOMs with a `protocol-removed` notification.

/connect ctype [source target host port protocol login password serverstring nativestring]
Establish a connection to a terminal or between two terminals.

At least one of the `source` and `target` arguments must be specified. If `source`, `ctype` must be either `ii` or `oo`. If target, `ctype` must be either `ii` or `oi`. Notifies its MOMs with a `connected` notification.

/disconnect ctype [source target host port protocol login password serverstring nativestring]
This is the mirror to the `connect` request. Notifies its MOMs with a `disconnected` notification.

3.6.2.3 Operational status operations

/controls
Returns an “control panel” for the monitor, written in HTML, Java, or Javascript. This panel may offer process-specific status information and controls. If no special controls are offered, the control panel might be a synonym for `ping`.

/
A synonym for `/controls`

/start [scope]
Begin processing input data streams. A no-op if the process is already started.

The unadorned form, which is synonymous with the `scope=all` operation, says that the main process and all subprocesses (recursively) should start processing their input data streams. The other variations on scope say to start only the main process, only its subprocesses (recursively), or an individual process.

Upon starting, each process sends a `started` notification to its MOMs. This applies to recursively started processes as well as the ones to which the initiating command was sent.

/stop [**scope**]

Stop processing input data streams. A no-op if the process is already stopped. This means to discard any inputs packets on the input queues and any new inputs subsequently received, and to discard any output packets on the output queues. Monitoring processes, when created, should be in the stopped state.

The scope options have the corresponding interpretations as with the `start` operation.

Upon stopping, each process sends a `stopped` notification to its MOMs. This applies to recursively stopped processes as well.

/pause [**scope**]

Temporarily halts processing of inputs. A no-op if the process is already paused. New packets received on the input terminals are received and queued. No output packets should be sent, but the output queues should be left intact.

The scope options have the corresponding interpretations as with the `start` operation.

Upon pausing, each process sends a `paused` notification to its MOMs. This applies to recursively paused processes as well.

/resume [**scope**]

Resumes processing of inputs. A no-op if the process is not paused. Same arguments as `pause`. Also notifies its MOMs with a `resumed` notification.

/init [**scope**]

The process resets all its internal parameters or data stores to their initial values, and flushes its input and output queues, to obtain something of the same effect as stopping and restarting, but without actually restarting the process. The process remains in the same operational state as before, whether stopped, started, or paused. Notifies its MOMs with an `initialized` notification.

/quit [**scope**]

The process ceases operation, and exits. Just before exiting, it notifies its MOMs with a `quitting` notification.

/checkpoint

This should instruct the process to save any of its valuable or state information, either as a precaution, or in anticipation of a forthcoming shutdown. Notifies its MOMs with a `checkpointed` notification.

/set-parameter name value

This instructs the process to set the operational parameter named by `name` to the value given by `value`. Notifies the MOM with a `parameter-set` notification.

3.6.2.4 MOM operations

/setid id

This instructs the process to record the supplied `id` value as its MOM-assigned identifier. Notifies its MOMs with a `identified` notification.

/setowner owner

This instructs the process to record the supplied `owner` value as its owner. Notifies its MOMs with a `owner-set` notification.

/setgroup group

This instructs the process to record the supplied `group` value as its owner group. Notifies its MOMs with a `group-set` notification.

/setpermissions permissions [target]

This instructs the process to record the supplied `permissions`. Notifies its MOMs with a `permissions-set` notification. A target specification indicates a terminal for which the permissions apply.

/add-mom host port [base name]

This adds an entry for a MOM to the processes' list of MOMs. Notifies the MOM with a `mom-added` notification.

/remove-mom host port [base name]

This removes an entry for a MOM to the processes' list of MOMs. Notifies the MOM with a `mom-removed` notification.

/replace-mom host port ohost oport [base obase]

This operation removes the entry for the MOM located at `host`, `port`, `base` and adds an entry for one at `ohost`, `oport`, `obase`. The combined operation is needed when the process has only one MOM or has a defunct MOM being replaced by a stepMOM, since after removing the earlier MOM there is no MOM to which a notification can be sent. Notifies both the new MOM and the old MOM with a `mom-replaced` notification, but does not signal an error if the old MOM notification fails.

/add-grandmom host port ohost oport [base obase]

This identifies a grandMOM for a given MOM. Notifies the MOM with a `grandmom-added` notification.

/remove-grandmom host port ohost oport [base obase]

This identifies a grandMOM for a given MOM. Notifies the MOM with a `grandmom-removed` notification.

Chapter 4

Control architecture

A MOM is a process that provides administrators and users with abilities to inspect, control, and modify monitoring processes and process-network connections. Each MOM is a monitoring process itself. It responds to all of the basic monitoring process operations that every monitoring process must handle, and responds to an additional range of commands (that is, process-specific commands) appropriate to its duties as a store of information about the monitoring network and as a tool for users in operating on the monitoring network.

Though a MOM is itself a monitoring process like those under its control, its different role is reflected in some differences from the usual properties of monitoring processes. The MOM, unlike most processes under its control, has no input or output terminals, only its control terminal. The MOM, like other nonatomic monitoring processes, has subprocesses, but routinely changes the set of these. In particular, each process initiated by the MOM is made one of its subprocesses; processes to which the MOM makes connections but not set up by it need not be viewed as subprocesses of the MOM. One views addition or removal of subprocesses and process-network connections as changing the current definition of the MOM subprocess network.

Different MOMs may control overlapping monitoring networks. A given MOM thus may have occasion to interact with other MOMs. The most immediate reason for such interaction is for sharing of information about jointly-administered monitoring processes. Perhaps the most important reason for inter-MOM interactions is sharing of monitoring knowledge, such as interchange of library updates and situational information. We leave the task of working out protocols for such interactions to future investigation.

A MOM records information associated with its operations in a persistent database to permit recovery and restarting of the monitoring system following disruptions. The MOM tables are distinct from the libraries of monitoring knowledge (for example, monitoring process descriptions) which generally record static abstract information rather than the particulars of active processes. Appendix A summarizes the main information tables and their fields, which cover the following subjects:

- Processes, including their terminals, subprocesses, connections, and port assignments;

- Connections,

- Displays,

- Users and groups,

- Host domains,

- System resources, including grandMOMs, other MOMs, process description and packet description libraries, script locations, and script server locations.

4.1 Human interface

While the MOM is a persistent process that, like other MAITA processes, receives commands via HTTP. While this architecture makes it possible to issue commands to the MOM directly through any web browser, manual issuance is too inconvenient for routine operation. To ease the exercise of control, human interaction with the MOM usually goes through HTML forms or Java applets specifically designed to make this interaction convenient when using a web browser. First among these is the MOM control panel, a graphical Java applet designed to provide a wide range of useful functionality to the developer and user of monitoring systems.

The MOM control panel issues commands directly to monitoring processes, but cannot receive communications instigated by these processes due to the security model prevailing in web browsers. The MOM itself handles communications from the processes being monitored; the control panel relies on the MOM from which it was spawned to send it updates and requests from these monitoring processes.

A single MOM can support one or more users at a time via multiple active control panels. More than one MOM may operate on the same process network or subnetwork. Each user sees only those processes allowed by the permissions on the processes, and can act on only those processes allowed by the process permissions.

4.1.1 Monitoring control displays

The MOM control panel provides displays for viewing the monitoring network, the status of monitoring processes, signals flowing through the network, and library entries. It also provides means for changing the structure of the monitoring network by adding or removing processes or data connections, and for calling up process-specific“control panels” for adjusting operational parameters and actions of the monitoring processes under its control.

4.1.1.1 Network display

The main MOM control-panel display features a set of operation buttons and pull-down menus, a textual alert area, and a textual help and response line, all organized around a pane providing a pictoral representation of the process-network structure. The user can zoom in, out, and about to examine portions of the network, select and deselect the processes and connections depicted in the network window, and use the operational buttons and menus to operate on the selected elements. Such operations include

- Calling up control panels for selected processes in separate windows,
- Issuing basic operating control commands to selected processes, such as starting, stopping, pausing, and resuming processing of inputs, reinitialization, quitting, checkpointing (saving of state), checking of operational status, and retrieval of information about the process,
- Creating data displays of information transmitted along connections in separate windows,
- Changing the network topology by adding or removing processes or connections,
- Changing the appearance or arrangement of the network elements in the network display window, including placement, sizes, fonts, and colors, with all these actions designed to conform to standard windowing conventions,
- Creating images or printed forms of portions of the network structure, and
- Retrieve, storing, and editing library specifications for monitoring processes or network fragments, using standard browsing and text editing environments and some OKBC browsing and editing applets developed elsewhere for performing these functions.

Each of the windows created by the MOM control panel is independently resizable and relocatable by the user.

A MOM provides means by which users or developers may change the network composition and structure by creating or destroying monitoring processes, connections, and subnetworks. In fact, the typical way of building up a monitoring network is to instantiate several processes or network fragments from a library, and then to connect these together. To facilitate this, the MOM provides a graphical interface depicting the network topology. The user can examine the graphical network display in various levels of detail. By selecting processes and connections depicted in the display, the user can indicate control operations, including destruction of the selected items. The user may also select terminals of displayed processes and create new data connections between these terminals.

4.1.1.2 Data displays

Both developers and users of monitoring systems need ways of telling what the network is doing, especially ways of observing the data streams coursing through the network. By selecting the icons representing different monitoring processes or terminals in the network display, the user can call up a variety of data-stream displays of common utility, including strip charts, multivariate displays, and maps, as described earlier.

The report type descriptions in the monitoring library provide default display types for each report type, and the monitoring process descriptions in the library can provide default display types for different processes and their terminals, overriding the default displays for the report types associated with the terminals.

Users may also define specialized display types and use these to override the default displays for specific monitoring processes.

4.1.2 System administration displays

In addition to the displays associated with controlling the active monitoring network, the MOM control panel provides access to various system administration displays. These displays permit authorized users to add or change information about users, groups, and ownership, as well as information about the location of knowledge bases, databases, and process creation scripts.

4.2 Constructing monitoring networks

Creation of new monitoring processes is done in either by instantiating library entries or by manually constructing networks. In either case, resources for the processes can be specified by the user or chosen automatically by the MOM.

4.2.1 Library network instantiation

The most common method for constructing monitoring networks is to use the MOM facilities to retrieve the description of a monitoring network fragment from the monitoring library. Such fragments appear as nonatomic process descriptions in the library. Once the user retrieves the appropriate network description, the user instructs the MOM to create an instance of the network. The MOM then creates the processes and monitoring network connections specified in the library entry, and recursively instantiates any nonatomic processes contained therein. The MOM then adds icons representing these new network fragments to the network display.

The usual sequence of operations is as follows:

1. The user identifies a monitoring process (which may consist of a network fragment) in one of the libraries provided by the MOM. The user may provide the address of a new library to the MOM, if the desired process is not contained in the libraries already known to the MOM. These operations are performed using the library browsing tools provided by the MOM.

2. The MOM retrieves the description of the process or network fragment from the library.

3. The MOM chooses a host on which to run the process. This may involve directions or advice from the user to help choose either the specific host or to choose the host domain in which to choose a host, if not from the MOM's default host domain.

4. The MOM examines its script location databases to determine the location of a script for the process on the selected host. This step and the preceding choice of host may be reversed, if not all scripts are located on all hosts.

5. The MOM then executes the creation script for the new monitoring process. The MOM passes a variety of information along to the new process through arguments to the process creation script, including the identifier (an integer) that the MOM assigns to the process, the location of the MOM's control terminal, the owner and permission information for the new process, etc. Some of this information may also be provided later by monitor control operations such as `setid`.

6. The MOM then repeats all these steps recursively for any subprocesses.

7. The MOM creates all the data connections specified by the description of the new process.

4.2.2 Manual process construction

The second method for constructing monitoring networks is to use the network display to create empty processes and to then add terminals, subprocesses, and connections to make the new process into an identifiable subnetwork. In this method, the subprocesses and connections may be new ones or may be existing ones. The user may then package up the constructed monitoring network as a new process type and save it away in the library to aid reuse and sharing.

4.2.3 Resource allocation

Without specific guidance from the user, a MOM will create a new process on a host of its choosing within a host domain under its control. Each MOM has a default host domain from which to choose hosts. Authorized users can change this domain and the tables of host domains. When creating an instance of a monitoring process with subprocesses, the subprocesses are created within the same domain as the parent process.

In the current implementation the MOM allocates processes to hosts either at random or guided by user specification. We expect future versions to incorporate more sophisticated resource allocation methods involving decision-theoretic models of the probability and utility the consequences of different possible allocations, so as to permit rational allocation decisions that reflect the utility of different monitoring tasks in the changing situation. Research in progress is aimed at monitoring the trustworthiness of different hosts for different purposes, and making rational allocations of processes to hosts in light of these judgments of trustworthiness.

At present, once a monitoring process is created on a host it stays there, but future extensions may provide facilities for relocating processes to other hosts. This might happen, for example, when a process or user detects that the process is not getting enough resources at its current location.

4.3 Maintaining monitoring processes

Each MOM performs a number of operations aimed at maintaining the functionality of the monitoring network for which it bears responsibility. Some of these operations are active, namely periodic checking to see that each process and data connection is working properly. Other operations are passive, namely receiving notifications from the processes themselves concerning changes in their operating status, acknowledgment of success in fulfilling MOM requests, and signals of errors that have occurred.

The MOM takes actions to correct failures that it has detected or which have been brought to its attention. If a monitoring process has failed, the MOM recreates the process and its process-network connections, first removing any inoperative relics if necessary, and using the stored process parameters to restore the process to its former behavior. The MOM may also notify human users when appropriate.

4.3.1 Process status polling

The MOM periodically polls the processes for which it bears responsibility to see that they are operating properly. Ordinarily this just amounts to issuing a `ping` command to each process periodically at a system-determined interval, such as every five minutes. In fact, the MOM provides a somewhat more flexible system for polling to permit each process to run a range of diagnostic tests at intervals chosen to make sense for the complexity of the tests. While one could demand that a process report on whether it is functioning properly in every response to a `ping` command, such checking be to costly to perform on demand in this way. Accordingly, we permit process descriptions to specify a set of diagnostic commands and information about the times and intervals at which these commands should be executed.

To perform this range of diagnostic tests, the MOM maintains a table of polling information that indicates, for each process, one or more polling operations. The polling table associates several items with each polling operation:

- A `check-time` and `check-interval`, only one of which need be specified. The MOM performs the indicated polling operation on the process at the `check-time`, if one is specified, and at a time of its own choosing if none is specified. The MOM then performs the polling operation at the frequency given by the `check-interval`, if one is specified. If no check interval is given, no checks are done after the first one. Values of check times are given in the format `HHMMSS DDCCYY TZ.Z`, where CC indicates the century and TZ.Z indicates the time zone relative to UTC. The time zone specification may be omitted, in which case the MOM interprets these times as relative to its own clock. Numerical values of the check interval are interpreted as numbers of minutes.

- A `restart-interval` which if specified indicates that the process should be restarted if it fails to respond to polling requests for the specified number of seconds. If no restart interval is given, the MOM will restart the process after any failure to respond.

- A `restart-condition`, which if specified identifies a result of the polling operation that calls for restarting the process.

- A `continue-condition`, which if specified identifies a result of the polling operation that calls for continued operation of the process, instructing the MOM to restart the process if any other result is returned. If both continue and restart conditions are specified, the restart condition takes precedence, that is, the process will be allowed to continue to operate as long as the polling response does not match the restart condition.

To perform a polling operation, the MOM sends a request to the process of the form

```
/<op>  time [host port base]
```

We have written this request in suggestive syntax as used in our operation descriptions, not in proper HTTP syntax. The idea is that the polling request supplies the time at which the poll is taken, and optionally also indicates the location of the control terminal of the MOM. The `op` used in the request is the polling operation specified in the polling table. For example, a simple polling request might specify `ping` in the polling table. In this case, the process receiving the `ping` request would ignore any arguments supplied, as none are necessary to respond to the request.

4.3.2 GrandMOMs

A grandMOM is simply a monitoring process that performs for the MOM itself the same periodic status-checking, complaint-handling, and restoration services that the MOM performs for other processes. The grandMOM typically runs on a different host from the MOM, in order to maximize system resilience. The MOM maintains its grandMOMs as ordinary subordinate monitoring processes.

When the grandMOM determines that the MOM has failed or disappeared, it kills the defunct MOM process if necessary and creates a new MOM to replace a defunct one. The grandMOM then notifies all the processes under the control of the former MOM of the identity of their new "stepMOM".

4.3.3 StepMOMs

A newly created stepMOM first checks the accuracy of the database entries listing monitoring processes. For each monitoring process listed, it checks to see if the process is still there. If the process is there and functioning, the stepMOM checks the accuracy of its information about the process. If the process is not functioning, the stepMOM recreates it as described above.

4.4 MOM control operations

4.4.1 Process network operations

While the command interface provided by the MOM client applet should suffice to handle most user requests, such requests are ultimately communicated to the MOM server via commands. The following requests cover process creation, creation of library entries for new process descriptions, and identification of libraries for use by the MOM.

/make-process [**type library host port base login password name domain**]

This asks the MOM to create a new process, either as a new type or as an instance of an existing library type. If the `type` argument supplies a value, the value should be a string giving the name used for the desired process type in either the default library of monitoring process descriptions or in the library indicated in the other arguments. A library may be indicated either by giving a string as the value of the `library` argument, which should identify some library already known to the MOM. Alternatively, the location and access information for the desired library should be provided using the `host`, `port`, `base`, `login`, and `password` arguments. If no `type` value is supplied, a process is created with a new type. In either case, if a `name` argument is provided, it provides a string that should be used as the name for the new process in the current network of subprocesses. The `domain` argument may be used to specify a desired host domain within which to create the process, if not the default host domain.

/make-terminal name type pid packet

This asks the MOM to create a new terminal for a process in the network. This is legal only if the process is not one created by instantiating a library entry, that is, only if the process is one created as a new type using the `make-process` operation without specifying a process library type. The `name` argument provides a string to use as the name of the new terminal. The `type` argument value should be either `input` or `output`. The `pid` argument should identify the process to which the terminal is to be added. This identification can be either the name by which the MOM knows the process, or the numeric identification number assigned by the MOM. The `packet` argument should name a packet type from the library to use for information transmitted through the terminal.

/add-subprocess process sub

This asks the MOM to make the process identified using the `sub` argument value into one of the subprocesses of the process identified using the `process` argument value. The values here are pathnames used by the MOM in identifying the processes.

/add-connection process conn

This asks the MOM to make the connection identified using the `conn` argument value into one of the internal connections of the process identified using the `process` argument.

/save name [scope]
Save a description of portions of the current network in the library using the value of `name` as the name of the new type of process. The `scope` argument supplies information about which portions of the network to save. The `scope` variable can take on the values `all`, `selected`, or a list of process names, indicating that the MOM should save all of the processes and connections, only the ones selected in the network display, or the ones specifically listed.

4.4.2 Process control operations

4.4.2.1 Process state operations

The MOM provides the service of storing operating parameters for those processes that lack persistent databases.

/record-parameter id host port time name value
This requests the process to record the value given by `value` for the operational parameter named by `name` of the process identified by `id`, `host`, and `port`.

Processes receiving requests for connections and other operations may wish to restrict access to specific users or groups. The MOM acts as the authority on such questions.

/verify-permission id host port time owner group permissions [login password ologin opassword]
This query asks the MOM to verify that the user identified in the `ologin` and `opassword` arguments is authorized to request an operation given the identity, group, and permissions stipulated in the `owner`, `group`, and `permissions` arguments. The optional `login` and `password` arguments are used to authenticate the process presenting the query, if necessary, though this can usually be determined by other means. The request results in a reply consisting of either the string `allowed` or `disallowed`.

4.4.2.2 Process notification operations

The MOM makes provision for reports from the processes under its supervision that indicate the completion of requests given to those processes by the MOM or other sources. The following list describes the basic set of notifications which processes send to the MOM. The MOM need not rely exclusively on such reports for keeping track of things, of course; in some cases, it may be preferable or necessary to query the processes about their status instead or as well.

/created id host port time
This reports that the process has been created operating at the given location.

/started id host port time
This reports that the process has started processing input data.

/stopped id host port time
This reports that the process has stopped processing input data at the indicated time.

/paused id host port time
This reports that the process has paused processing input data.

/resumed id host port time
This reports that the process has resumed processing input data following a pause.

/initialized id host port time
This reports that the process has reinitialized its internal state.

/quitting id host port time
This reports that the process is ceasing operations and exiting.

/checkpointed id host port time
This reports that the process has completed a checkpointing operation.

/identified id host port time
This reports that the process has recorded a MOM identifier.

/protocol-added id host port time target protocol version login password duration
This reports the addition of a listener for a protocol on a terminal.

/protocol-removed id host port time target protocol version
This reports the termination of a listener for a protocol on a terminal.

/connected id host port time ctype source target protocol
This reports that the process has established a connection.

/disconnected id host port time ctype source target protocol
This reports that the process has eliminated a connection.

/owner-set id host port time
This reports that the process has recorded a new owner.

/group-set id host port time
This reports that the process has recorded a new group.

/permissions-set id host port time
This reports that the process has recorded new permissions.

/parameter-set id host port time name value
This reports that the process has set the operational parameter named by `name` to the value given by `value`.

4.4.2.3 Process error operations

Monitoring processes send signals to their MOMs when they encounter errors, either in their own functioning or in communicating with other processes, and to the appropriate grandMOM when they encounter problems in communicating with a particular MOM.

We will elaborate error sources and signal types in a future version of this specification.

4.4.3 Resource management operations

4.4.3.1 Library resource operations

/register-library host port [base login password ltype]
This asks the MOM to register the location of a library in its library registry. The `ltype` argument takes on as values either `process` or `packet`, and indicates the type of library being registered. The other arguments identify the location and access information for the new library.

/unregister-library host port [base login password ltype]
This asks the MOM to remove a library from its registry. The arguments are parallel to those of the `register-library` operation.

/register-script-server host port [base login password server-type]
This asks the MOM to register the location of a script server. The `server-type` argument takes on the values `web` or `scripts`, indicating that the server is either a full web server (the default) or a specialized script server. The other arguments identify the location and access information for the server.

/unregister-script host port [**base**]
This asks the MOM to remove a script from its script registry.

/register-script type host port [**base login password script-type**]
This asks the MOM to register the location of a script in its script registry. The `type` argument provides a string naming a type in the library

/unregister-script mp-type host port [**base login password script-type**]
This asks the MOM to remove a script from its script registry.

4.4.3.2 Domain resource operations

/domains [**user group**]
This asks the MOM to return a table of the defined host domains in the format:

```
<n> host domains
<name-1> <user-1> <group-1>
...
<name-n> <user-n> <group-n>
```

If `user` or `group` arguments are given, the table returned is restricted to those host domains with matching user or groups.

/register-domain name [**user group**]
This asks the MOM to register a new host domain, with owners as indicated by the optional `user` and `group` string arguments.

/unregister-domain name
This asks the MOM to remove a host domain from its registry.

/domain-contents [**domain**]
This asks the MOM to return a record of the contents of host domains. If no domain is specified as an argument, the contents of all host domains are returned. The contents of each host domain content report is given by the format:

```
Host domain <domain-name>
<n1> included hosts
    <host-1>
    ...
    <host-n1>
<n2> included subdomains
    <domain-1>
    ...
    <domain-n2>
<n3> excluded hosts
    <host-1>
    ...
    <host-n3>
<n4> excluded subdomains
    <domain-1>
    ...
    <domain-n4>
<n5> hosts
<host-1>
...
<host-n5>
```

/register-host domain host [**exclude**]
This asks the MOM to add the indicated host to the indicated domain. The host is included in the domain if no `exclude` argument is given, or if the value of that argument is ”false”.

/unregister-host domain host [**exclude**]
This asks the MOM to remove a host from the indicated domain.

/register-subdomain domain subdomain [**exclude**]
This asks the MOM to add the indicated subdomain to the indicated domain. The subdomain is included in the domain if no `exclude` argument is given, or if the value of that argument is ”false”.

/unregister-subdomain domain subdomain [**exclude**]
This asks the MOM to remove a subdomain from the indicated domain.

/default-domain domain
This asks the MOM to report the identity of its default domain. The format of the report returned is

```
MOM <host> <port> <base>
Default domain <domain>
```

/set-default-domain domain
This asks the MOM to change its default domain to the indicated domain.

4.4.3.3 GrandMOM and sister MOM operations

At present, a MOM responds to only two commands regarding grandmoms: one to register a grandmom with the MOM, and one to unregister a grandmom. Though the MOM itself will ordinarily set up a grandmom to monitor the MOM's functioning, we provide these operations to formalize the registration of grandmoms. This formalization permits additional grandmoms to be registered, even if created by users apart from the MOM they monitor.

/register-grandmom host port [**base login password**]
This asks the MOM to register the location of a grandmom.

/unregister-grandmom host port [**base login password**]
This asks the MOM to remove a grandmom from its registry.

At present, we provide operations only to let a MOM know about other MOMs operating in its “vicinity”. We do not presume a MOM needs to know about any other MOMs, or that it can know about all other MOMs, but the operations below let other MOMs register with each other, perhaps on instruction of their users.

/register-mom host port [**base login password name**]
This asks the MOM to register the location of another MOM. The `name` argument is used to convey a string denoting the name the MOM being registered uses to denote for itself.

/unregister-mom host port [**base login password**]
This asks the MOM to remove a MOM from its registry.

4.4.4 Display operations

The current implementation of the MOM provides displays only through the medium of the MOM control panel, in which requests for displays create subthreads of the control panel applet that in turn construct the display windows.

Future extensions of the implementation may also provide means for requesting displays directly from the MOM server. In this case, the MOM server would determine that the request was coming from a source other than the MOM client applet, and would respond with an applet specifically constructed to produce the desired display. Providing such a facility requires developing an argument syntax for specifying a selected portion of the network or particular data streams.

/display termlist [**display-type**]

This asks the MOM to create a display of some form, where the termlist argument provides a list of terminal identifiers and the display-type argument, if given, provides a display type indicator taken from the set `strip`, `multi-strip` (the default), `2d`, `text`, or the name of a display type description in the library.

/print [**image-type**]

Produce a depiction of the current network graph to some printer or stream in the format specified by the `image-type` argument, which should allow values including `gif`, `jpeg`, and `ps`, where `ps` indicates a postscript format image.

Chapter 5

Conclusion

The MAITA system provides tools for constructing, maintaining, and modifying distributed monitoring systems. It allows open integration with external systems and special-purpose monitor processes. It exploits a knowledge base of general and specific world knowledge in organizing monitoring knowledge and monitoring activities.

Future documents will address the structure of the monitoring knowledge base and resource allocation issues, both of which remain under development.

Appendix A

Information Tables

The MAITA system stores some basic information about the structure and status of monitoring networks in a persistent database in order to maintain stable operation. This appendix lists the tables, their contents, and their purposes.

A.1 Process tables

The process tables record information about the location, status, and permissions of active processes. Without this information, the MOM has no way of communicating with the processes.

Processes

The table of processes tells how to locate the process control terminals, and records some summary information about the status of the process. The fields and their types are as follows:

- `process-id`: an integer identifier assigned by the MOM.
- `process-type`: a string giving the name of the library type from which the process was instantiated.
- `short-name`: a string giving a short form of the name for use in graphical displays of the monitoring network. The short name may be derived from the process type and ID, or through some other scheme.
- `host`: a string giving either an alphanumeric domain name or a numeric IP address of the host on which the process runs
- `port`: an integer giving the port on which the control terminal listens
- `base`: a string giving a base URL component for the control terminal, if any
- `op-status`: a string giving the operational status of the process
- `fun-status`: a string giving the functional status of the process
- `creation-time`: a string giving the time at which the process was created
- `start-time`: a string giving the time at which the process was started
- `pause-time`: a string giving the time at which the process was paused
- `init-time`: a string giving the time at which the process was last initialized. This should be the same time as the creation time for processes which have not been explicitly reinitialized.
- `owner`: a string giving the user name of the owner of the process
- `group`: a string giving the name of the group owning the process
- `permissions`: a string giving the permissions of the process for owner, group, and other

Terminals

This table associates terminals with the process, giving their internal identifiers, packet type, and recording permission information for terminal operations.

- `process-id`: an integer, referring to the process ID in the table of processes
- `terminal-type`: a string, identifying the terminal as `input` or `output`
- `terminal-id`: an integer, assigned by the MOM to the terminal
- `name`: a string, giving the name of the terminal as found in the library description
- `packet-type`: a string, giving the name of the packet type of information transmitted through the terminal
- `owner`: a string giving the user name of the owner of the terminal (which may be different from that of the process of which the terminal is a part)
- `group`: a string giving the name of the group owning the terminal (which may be different from that of the process of which the terminal is a part)
- `permissions`: a string giving the permissions of the terminal for owner, group, and other

Subprocesses

This table associates processes with their subprocesses.

- `parent-id`: an integer, giving the ID of the parent process
- `child-id`: an integer, giving the ID of the child process

Process-connections

This table associates processes with their integral connections, for example, connections between their subprocesses as specified by the library description of the process. *Ad hoc* connections specified by the user at runtime would be entered in this table as elements of the top-level network being managed by the MOM.

- `process-id`: an integer, giving the ID of a process having integral process-network connections
- `connection-id`: an integer, giving the ID of a connection from the table of connections

Ports

This table records the port numbers used by terminals in listening for connections according to the different protocols.

- `process-id`: an integer, giving the ID of the terminal's process
- `terminal-type`: a string, telling whether the terminal is an input or output terminal
- `terminal-id`: an integer, giving the ID of the terminal
- `protocol`: a string, giving the type of protocol served by the port
- `version`: a string, giving the version of the protocol being served
- `host`: a string, giving the domain name or IP address of the host on which the listener runs
- `port`: an integer, giving the port number on which the listener listens
- `base`: a string, giving a URL base for the listener, if any
- `duration`: a string, giving the duration over which the terminal should exist. Possible values are `once`, `indefinitely`, `until-time`, and `until-successful`
- `until-time`: a string, giving a time at which the port should be closed when the `duration` field has the value `until-time`.

Polling

This table records information about how and when to poll monitoring processes.

- `process-id`: an integer, giving the ID of the process
- `operation`: a string, to be used as the operation in the HTTP command given to the process
- `check-time`: a string, giving the time at which the polling operation should be performed
- `check-interval`: a number, giving the number of minutes between successive checks
- `restart-interval`: a number, giving the number of seconds of failure to respond after which the process should be restarted
- `restart-condition`: a string, giving a result which, if obtained as the result of the polling operation, indicates the process should be restarted
- `continue-condition`: a string, giving a result which, if obtained as the result of the polling operation, indicates the process should be allowed to continue operating

A.2 Connection tables

The connection tables record each process network connection, giving the identities of the terminals or other destinations so connected.

Connections

This table associates identifying numbers with each connection, and gives the basic information about the packet structure of information crossing the connection, as well as the permission information for who may modify the connection.

- `connection-id`: an integer, assigned by the MOM to the connection as an internal identifier
- `protocol`: a string, giving the protocol used for transmitting data across the connection
- `packet-type`: a string, giving the name of the packet type of information on the connection
- `parent`: an integer, giving the process ID of the process in which this connection exists, or 0 if the MOM alone has control of the connection
- `owner`: a string, giving the name of the user owning the connection
- `group`: a string, giving the name of the group owning the connection
- `permissions`: a string, giving the permissions for the owner, group, and others

Connection-sides

This table provides information about the sources and targets of information flowing into a connection. We refer to each of the source and target as a “side” of the connection.

- `connection-id`: an integer, corresponding to a connection ID number from the connections table
- `side`: a string, indicating either `source` or `target`
- `connection-type`: a string, either `terminal` or `external`, indicating whether the connection side is a process terminal or an external location
- `process-id`: an integer, identifying a process for internal connection sides
- `terminal-id`: an integer, identifying a terminal of the process, for internal connection sides
- `host`: a string, giving the domain name or IP address on which the listener for connections to the connection side runs

- port: an integer, giving the port number for the listener for connections to the connection side
- base: a string, giving the URL base for the listener for connections to the connection side
- login: a string, giving the user name for accessing the connection side listener, if any
- password: a string, giving the password for accessing the connection side listener, if any

A.3 Display tables

Control panels

This table gives information about MOM control panels operating through the MOM, namely the information about the location of the control panel and the user logged in through it. A single MOM may support several users through several control panels. Each control panel may generate several displays.

- id: an integer, used to identify the control panel
- host: a string, giving a domain name or IP address for the host providing the control panel
- port: an integer, giving the port number for the control panel listener
- base: a string, giving the URL base of the control panel request, if any
- uname: a string, giving the login name for the user logged in through the control panel
- password: a string, giving the password for used for login

Displays

This table gives information about displays being operated by the MOM, namely the information about the location of the display and who is authorized to change it.

- id: an integer, used to identify the display
- host: a string, giving a domain name or IP address for the host providing the display
- port: an integer, giving the port number for the display server
- base: a string, giving the URL base of the display request, if any
- uname: a string, giving the user name for display requests, if needed
- password: a string, giving the password for display requests, if needed
- owner: a string, identifying the user that owns the display
- group: a string, identifying the group owing the display

Containers

This table describes containers, which are essentially HTML-like tables in the cells of which one places panes and other items.

- id: an integer, used to identify the container
- display: an integer, corresponding to the display ID of the display operating the container
- left-visible: a boolean, T means make the left side visible
- right-visible: a boolean, T means make the right side visible
- top-visible: a boolean, T means make the top side visible
- bottom-visible: a boolean, T means make the bottom side visible
- resizable: a boolean, T if the container as a whole is resizable
- pausable: a boolean, T if all the signals are pausable.

- `ll-vertical`: a number, giving the vertical location of the lower left corner of the container
- `ll-horizontal`: a number, giving the horizontal location of the lower left corner of the container
- `height`: an integer, giving the vertical size of the container in pixels
- `width`: an integer, giving the horizontal size of the container in pixels
- `background-type`: a string, one of color or transparent
- `background-color`: a string, indicating the background color, if any

Panes

This table describes panes that constitute the heart of any display.

- `id`: an integer, used to identify the pane
- `type`: a string, one of 1D, 2D, map, and text
- `container`: an integer, corresponding to the container ID of the container containing the pane, if any
- `background-type`: a string, one of image, color, or transparent
- `background-image-resizable`: a boolean, T if the background image is resizable
- `background-image-url`: a string, giving a URL for a background image, if there is one
- `background-color`: a string, indicating the background color, if any
- `v-resizable`: a boolean, T if the pane is resizable in the vertical direction
- `h-resizable`: a boolean, T if the pane is resizable in the horizontal direction
- `left`: an integer, corresponding to a side ID for the left side of the pane
- `right`: an integer, corresponding to a side ID for the right side of the pane
- `top`: an integer, corresponding to a side ID for the top side of the pane
- `bottom`: an integer, corresponding to a side ID for the bottom side of the pane
- `ll-vertical`: a number, giving the vertical location of the lower left corner of the pane within the coordinate system defined by the pane container
- `ll-horizontal`: a number, giving the horizontal location of the lower left corner of the pane within the coordinate system defined by the pane container
- `height`: an integer, giving the vertical size of the pane in pixels
- `width`: an integer, giving the horizontal size of the pane in pixels

Sides

This table records information about intervals used to mark off the sides of panes.

- `id`: an integer, used to identify the side
- `interval`: an integer, corresponding to an interval ID
- `location`: a string, one of top, bottom, left, and right. Top isn't used at present, but is included here for future expansion.
- `label`: an integer, corresponding to a label ID with which to label the side

Labels

This table gives content and formatting information on labels used in displays.

- `id`: an integer, used to identify the label
- `content`: a string, containing the text to be used as the label

- `font`: a string, identifying the font in which to display the label
- `size`: an integer, identifying the font size in which to display the label
- `color`: a string, identifying the color in which to display the label
- `format`: a string, used to indicate the format desired for the content. The strings will vary widely with the type of information being displayed. Numerical values will typically use C/Java-style formatting indicators; time values will be specified using a formatting vocabulary like that used in Common Lisp.
- `placement`: a string, one of max, min, or midpoint
- `orientation`: a string, one of horizontal or vertical
- `inversion`: a boolean, T meaning invert the

Scales

This table gives information about dimensions (time, space, etc.) and the scales used to measure elements of the dimensions.

- `id`: an integer, used to identify the scale
- `dimension`: a string, giving the name of the dimension being measured (for example, time)
- `unit`: a string, giving the name of the unit of measurement (for example, seconds)
- `numeric`: a boolean, T meaning measurements are numbers
- `max`: a string, giving the maximum possible value of measurements (empty if infinite)
- `min`: a string, giving the minimum possible value of measurements (empty if infinite)

Intervals

This table gives information about an interval of values (from some scale) displayed along a side of some pane.

- `id`: an integer, used to identify the interval
- `scale`: an integer, corresponding to a scale ID from the scales table
- `floating`: a boolean, F for intervals with fixed endpoints (the default), T for intervals with moving endpoints (for example, time windows). Max and min values below make sense for fixed intervals only, while the size value makes sense for floating intervals
- `max`: a string, giving the maximum value of measurements in the interval (empty if infinite)
- `min`: a string, giving the minimum value of measurements in the interval (empty if infinite)
- `size`: a number, giving the size of the interval for floating intervals (size means number of values in the interval for non-numeric scales)
- `display-division`: a boolean, T by default, indicating whether division (tick) marks should be displayed or not
- `division-size`: a number, giving the size of the divisions in scale units
- `division-origin`: a number, giving some point on the scale that should have a division mark; marks should be placed at greater and lesser values on the scale that differ from the origin by a multiple of the division-size.
- `label-division`: a boolean, T if the division marks should be labeled with the values they indicate
- `division-label`: an integer, indicating an entry in the labels table. The content of this entry is ignored, only the formatting information is used for division labels.

- `display-endpoints`: a boolean, T if the min and max values should be displayed independent of any markers and division points
- `display-markers`: a boolean, T by default, indicating whether to display any markers specified in the markers table
- `resizable`: a boolean, T if the interval may be resized along the indicated scale

Markers

This table records stipulated markers within intervals.

- `interval`: an integer, corresponding to an interval ID
- `value`: a string, corresponding to a value on the scale of the interval (but not necessarily within the region denoted by the interval)
- `label`: an integer, corresponding to a label ID for the label to mark the value

Signals

This table gives information about the signals displayed in the panes.

- `id`: an integer, used to identify the signal
- `name`: a string, used to describe the signal. By default, constructed using the h-label, v-label, and source name below
- `source`: an integer, corresponding to a source ID
- `color`: a string, giving a color name or number
- `v-param-name`: a string, corresponding to the name of a packet field in the source packet, indicating the parameter to be used to provide a vertical coordinate
- `h-param-name`: a string, corresponding to the name of a packet field in the source packet, indicating the parameter to be used to provide a horizontal coordinate
- `h-label`: an integer, corresponding to a label ID used to label the vertical parameter if needed
- `v-label`: an integer, corresponding to a label ID used to label the horizontal parameter if needed
- `v-side`: an integer, corresponding to the side ID of the side against which the vertical parameter is plotted
- `h-side`: an integer, corresponding to the side ID of the side against which the horizontal parameter is plotted
- `style`: a string, one of curve, step or point. Curve means to draw continuous curves between successive points; step means to draw constant curves from each point until the next, with jumps from one value to the next; and point means to plot each point separately, with no connecting curves between points.

Pane-Signals

This table tells which signals appear on which panes. The same signal may be displayed on more than one pane, and a single pane may display more than one signal.

- `pane`: an integer, corresponding to a pane ID
- `signal`: an integer, corresponding to a signal ID

Sources

This table gives information about the source streams from which displayed signals should be drawn.

- `id`: an integer, used to identify the source

- `host`: a string, giving a domain name or IP address for the host providing the source stream
- `port`: an integer, giving the port number for connecting to the source
- `base`: a string, giving the URL base of the connection request, if any
- `uname`: a string, giving the user name for connection requests, if needed
- `password`: a string, giving the password for connection requests, if needed
- `terminal`: an integer, giving the terminal ID of the terminal providing the stream
- `packet`: a string, giving the name of the packet type used to encode the information
- `protocol`: a string, giving the protocol by which the stream is transmitted from the connection port
- `version`: a string, giving the protocol version, if any

A.4 User tables

Users

This table gives the defining information for individual users.

- `user-id`: an integer, assigned by the MOM as an identifier for the user
- `login-name`: a string, giving the login name by which the user identifies himself
- `password`: a string, giving the user's password for obtaining permission to perform operations
- `email`: a string, giving an email address, of the form `<user-name>`@`<host>`
- `firstname`: a string, giving the first name of the user
- `lastname`: a string, giving the last name of the user
- `middlenames`: a string, giving the middle name or names of the user
- `last-login-time`: a string, giving the time at which the user last accessed the system

User-history

This table gives histories of actions by the users.

- `user-id`: an integer, giving the ID of a user
- `time`: a string, giving the time at which the user performed an operation
- `operation`: a string, giving the operation performed by the user at the indicated time

Groups

This table gives the defining information for user groups.

- `group-id`: an integer, assigned by the MOM as an identifier for the group
- `name`: a string, by which users identify the group
- `fullname`: a string, giving a longer description of the composition or role of the group

Group-members

This table indicates which users are members of which groups.

- `group-id`: an integer, giving the ID of a group
- `user-id`: an integer, giving the ID of a user who is a member of the group

A.5 Host domain tables

MOMs choose hosts on which to run processes from within host domains (not to be confused with internet domains) defined through the following set of tables.

Domains

This table lists the domains defined in the system.

- `name`: a string, giving the name of the domain
- `user`: a integer, giving the ID of a user owning the domain
- `group`: an integer, giving the ID of a group owning the domain

Hosts

This table lists individual hosts included in or excluded from domains.

- `domain`: a string, giving the name of a domain
- `host`: a string, giving the domain name or IP address of the host to be included or excluded from the indicated domain
- `excluded`: a boolean, true if the host should be excluded instead of included

Subdomains

This table lists subdomains to be included in or excluded from domains.

- `domain`: a string, giving the name of the parent domain
- `child`: a string, giving the name of the subdomain
- `excluded`: a boolean, true if the child should be excluded instead of included

Default domains

This table associates domains with MOMs, which are identified by their locations.

- `host`: a string, giving the domain name or IP address of the host on which the MOM runs
- `port`: an integer, giving the port number on which the MOM's control terminal listens
- `base`: a string, giving the URL base for requests to the MOM
- `domain`: a string, giving the name of the default domain for the MOM

A.6 System resource tables

Grandmoms

This table lists processes acting as grandmoms for the MOM, that is, as processes which monitor the MOMs own functioning and restart the MOM if need be. Usually there is only one grandmom, and it is an ordinary subprocess managed and maintained by the MOM itself.

- `host`: a string, giving the domain name or IP address of the host on which the grandmom runs
- `port`: an integer, giving the port number on which the grandmom's control terminal listens
- `base`: a string, giving the URL base for requests to the grandmom
- `login`: a string, giving the user name for accessing the grandmom
- `password`: a string, giving the password for accessing the grandmom

Other MOMs

This table lists other known MOMs, giving their names (if any), locations.

- `host`: a string, giving the domain name or IP address of the host on which the other MOM runs
- `port`: an integer, giving the port number on which the other MOM's control terminal listens
- `base`: a string, giving the URL base for requests to the other MOM
- `login`: a string, giving the user name for accessing the other MOM
- `password`: a string, giving the password for accessing the other MOM
- `name`: a string, giving a name by which the other MOM is known
- `reconciliation-time`: a string, giving the last time at which the MOM and the other MOM reconciled their directory information
- `last-status-time`: a string, giving the time at which the indicated MOM was last checked to see if it was still operational

Libraries

This table lists locations for one or more libraries of monitoring process descriptions, packet types, etc.

- `libtype`: a string, giving the type of library (process description, packets, etc.)
- `host`: a string, giving the domain name or IP address of the host on which the knowledge-base server runs
- `port`: an integer, giving the port number on which the knowledge-base server listens
- `base`: a string, giving the URL base for requests to the knowledge-base server
- `login`: a string, giving the user name by which to access the knowledge base
- `password`: a string, giving the password by which to access the knowledge base
- `kbtype`: a string, giving the type of knowledge base (CYC, Loom, etc.)
- `protocol`: a string, indicating the protocol by which to access the knowledge base (either OKBC or one of the kbtype values, indicating a knowledge-base-specific protocol)
- `session`: a string, indicating a session ID for an OKBC session
- `duration`: a string, indicating a session duration for an OKBC session
- `key`: a string, indicating a session key returned by an OKBC session
- `kb-library`: a string, indicating a library within an OKBC knowledge base
- `order`: an integer, indicating the order in which libraries should be used. This specification does not rule out ties, but does presume that libraries of order 0 are used first, then order 1, etc.

Scripts

This table tells where executable versions of that type of process may be found within the domain of hosts available to the MOM.

- `mp-type`: a string, giving the name of a type of monitoring process in the monitoring library
- `host`: a string, giving the domain name or IP address of a host identifier in the domain of the MOM
- `script`: a string, giving a pathname that identifies the desired script to the script server on that host

Script servers

This table tells where script servers are located on hosts available to the MOM.

- `host`: a string, giving the domain name or IP address of the host on which the script server runs
- `port`: an integer, giving the port on which the script server listens
- `base`: a string, giving the URL base for requests to the script server
- `login`: a string, giving the user name by which to access the script server
- `password`: a string, giving the password by which to access the script server
- `server-type`: a string, indicating whether the server is a full web server (`web`) or a specialize MAITA process creation server (`maita`)

Bibliography

[1] J. A. Breuker and W. Van de Velde, editors. *The CommonKADS Library for Expertise Modelling*. IOS Press, Amsterdam, 1994.

[2] J. Doyle, Y. Shoham, and M. P. Wellman. A logic of relative desire (preliminary report). In Z. W. Ras and M. Zemankova, editors, *Methodologies for Intelligent Systems, 6*, volume 542 of *Lecture Notes in Artificial Intelligence*, pages 16–31, Berlin, Oct. 1991. Springer-Verlag.

[3] J. Doyle and M. P. Wellman. Representing preferences as *ceteris paribus* comparatives. In S. Hanks, S. Russell, and M. P. Wellman, editors, *Proceedings of the AAAI Spring Symposium on Decision-Theoretic Planning*, 1994.

[4] J. Fackler, I. J. Haimowitz, and I. S. Kohane. Knowledge-based data display using trendx. In *AAAI Spring Symposium: Interpreting Clinical Data*, Palo Alto, 1994. AAAI Press.

[5] P. Haddawy and S. Hanks. Representations for decision-theoretic planning: Utility functions for deadline goals. In B. Nebel, C. Rich, and W. Swartout, editors, *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, pages 71–82, San Mateo, CA, 1992. Morgan Kaufmann.

[6] I. J. Haimowitz and I. S. Kohane. Automated trend detection with alternate temporal hypotheses. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 146–151, Chambery, France, 1993.

[7] I. J. Haimowitz and I. S. Kohane. An epistemology for clinically significant trends. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 176–181, Washington, DC, 1993.

[8] B. Hayes-Roth, S. Uckun, J. E. Larsson, J. Drakopoulos, D. Gaba, J. Barr, and J. Chien. Guardian: An experimental system for intelligent ICU monitoring. In *Symposium on Computer Applications in Medical Care*, Washington, DC, 1994.

[9] B. Hayes-Roth, R. Washington, D. Ash, R. Hewett, A. Collinot, A. Vina, and A. Seiver. Guardian: A prototype intelligent agent for intensive-care monitoring. *Journal of AI in Medicine*, 4:165–185, 1992.

[10] I. Kohane and I. Haimowitz. Hypothesis-driven data abstraction. In *Symposium on Computer Applications in Medical Care*, Washington, DC, 1993.

[11] I. S. Kohane. Temporal reasoning in medical expert systems. In R. Salamon, B. Blum, and M. Jørgensen, editors, *MEDINFO 86: Proceedings of the Fifth Conference on Medical Informatics*, pages 170–174, Washington, Oct. 1986. North-Holland.

[12] I. S. Kohane. Temporal reasoning in medical expert systems. TR 389, Massachusetts Institute of Technology, Laboratory for Computer Science, 545 Technology Square, Cambridge, MA, 02139, Apr. 1987.

[13] I. S. Kohane and I. J. Haimowitz. Encoding patterns of growth to automate detection and diagnosis of abnormal growth patterns. *Pediatric Research*, 33:119A, 1993.

[14] S. M. Ornstein, D. R. Garr, R. G. Jenkins, P. F. Rust, and A. Arnon. Computer-generated physician and patient reminders. *Journal of Family Practice*, 32:82–90, 1991.

[15] D. M. Rind, C. Safran, R. S. Phillips, W. V. Slack, D. R. Calkins, T. L. Delbanco, and H. L. Bleich. The effect of computer-based reminders on the management of hospitalized patients with worsening renal function. In P. Claytons, editor, *Proceedings Symposium Computer Applications in Medical Care*, pages 28–32, Washington, DC, 1991. McGraw-Hill.

[16] D. M. Rind, C. Safran, R. S. Phillips, Q. Wang, D. R. Calkins, T. L. Delbanco, H. L. Bleich, and W. V. Slack. Effect of computer-based alerts on the treatment and outcomes of hospitalized patients. *Archives of Internal Medicine*, 154:1511–1517, 1994.

[17] M. P. Wellman and J. Doyle. Preferential semantics for goals. In *Proceedings of the National Conference on Artificial Intelligence*, pages 698–703, 1991.

[18] M. P. Wellman and J. Doyle. Modular utility representation for decision-theoretic planning. In *Proceedings of the First International Conference on AI Planning Systems*, 1992.