ESPRIT Project No. 25 338

Work package G

Service Deployment

Design

ID:	WP G – Design V. 1.1	Date:	28.04.1998
Author(s):	L. Amsaleg, M. Billot, P. Couderc, V. Issarny, A.-M. Kermarrec, M. Le Nouy, J.-P. Routeau	Status:	Draft
Reviewer(s):	E. Triep	Distribution:	Deliverable


FollowMe

Change History

Document Code	Change Description	Author	Date
WPG Design V1.0	First version of document. No changes.	INRIA TCM.	23.03.98
WPG Design V1.1	Incorporation of the internal review suggestions	INRIA TCM.	28.04.98

1 Introduction

The architecture of FollowMe raises new issues related to distributed services. New opportunities are based on agent technology which provides mobile code. In work package G, we are investigating some methods for taking advantage of these new opportunities in order to provide load-balancing tools. Our intention is not to cover the wide spectrum of load-balancing tools. Rather, we focus on a set of tools that could help the deployment of a large scale, widely distributed application over th FollowMe infrastructure. For example, we consider the deployment of Etel++ over the architecture of FollowMe. This application, described in [DJ2] includes distributed computing, extensive usage of storage and large data transfer. Therefore, it requires efficient load-balancing policies relying on accurate monitoring tools.

The main goals of work package G are therefore as follows:

- Allowing applications to monitor the performance of hosts. As an example, this monitoring may deliver numbers reflecting the resource consumption of an agent or the available bandwidth between two hosts. Monitoring tools should also provide a convenient way to extract useful information of the large amount of numbers the monitors deliver. These tools exhibit tendencies, meaningful numbers (average, deviation, mean-square…) and also predictive values.
- Implementing a specific load-balancing policy dedicated to Etel++. This policy is based on both monitoring tools an data mining techniques.

Section 2 provides a brief overview of the requirements of work package G. Section 3 summarizes the state of the art concerning the resource monitoring and data mining which both appear as key features for improvement of load-balancing in FollowMe. Section 4 describes the proposed design. Section 5 describes some use cases relevant to ETEL++ deployment. Finally, Section 6 proposes a short status of work package G and the ongoing work.

2 Requirements

We provide in this section a short description of requirements for work package G. Section 2.1 gives an overview of these requirements. Section 2.2 describes the purposes of monitoring tools. Section 2.3 describes the key features of load-balancing policy.

2.1 Overview

Work package G can be viewed as a toolbox that may help high-level applications to take performance-related decisions, such as determining a good way to balance the load they impose on the system. In a nutshell, this work package provides *mechanisms* for evaluating the usage of resources, that is, it provides a set of monitoring tools. Application may use these tools to enforce a *policy* achieving a particular goal, load balancing being a possibility. This separation between mechanisms (that are part of this work package) and the policies (that are part of other work packages) is important. Therefore, the tools described here have to be generic enough for being possibly used in different contexts and for being included in various policies.

While having general-purpose tools is a requirement, it should be apparent that each policy assembles some of these tools in a very specific way. Therefore, two policies are likely to be made of different tools belonging to this work package, and furthermore, each may use the tools differently. In general, a policy that uses the tools of this work package can be conceptually split in three major parts: (i) the tools themselves, (ii) a list of requirements that the policy has to enforce and (iii) an "engine" that uses the data returned by the tools to actually enforce the requirements.

The purpose of this document is to present two of the three parts mentioned above, namely the tools themselves and a way to use these tools to enforce a specific policy. This policy is dedicated to ETEL++, and has been designed to enforce the Quality of Service requirements of this pilot application. (Note that INRIA is responsible for both this work package and the pilot application 2.)

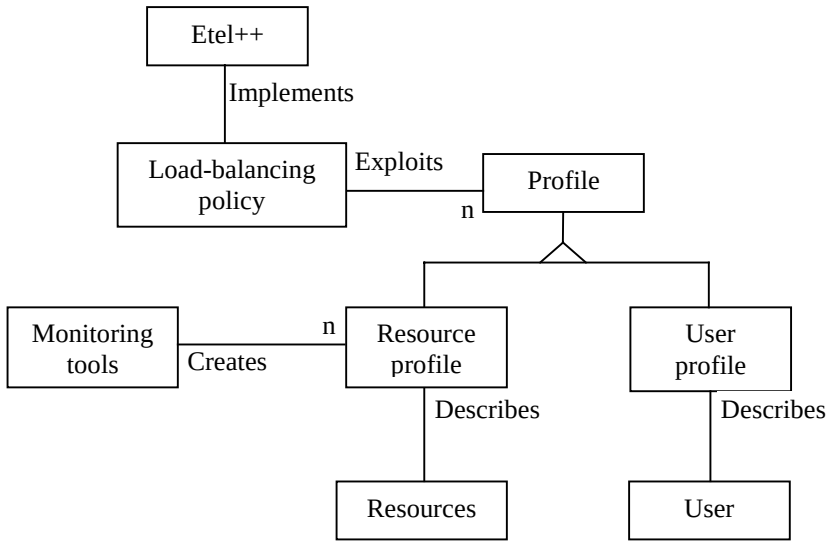

Figure 1 : Load-balancing tools

The Figure 1 helps in getting a global view of what this work package is made of. The three parts mentioned above are represented. The first part – the tools themselves— can conceptually be viewed as profiles describing the usage of resources, this usage being maintained by the monitoring tools. Profiles may for example describe the availability of resources in terms of percentage of free disk space, CPU load or Network bandwidth. The second part –the list of constraints to eventually enforce—can be conceptually viewed as a user profile. A constraint may for example be that the user needs to browse his personalized newspaper every day around 9am. Finally, the third part exploits both the resources profiles and the user profiles to devise an appropriate behavior enforcing the constraints. A possible behavior may for example be building the personalized newspaper overnight, then shipping it to a computer close to the location of the user a couple of hours before 9am.

This work package provides a convenient notion of resource profile and also a way to exploit the profile concept within a load-balancing policy dedicated to ETEL++. The proposed architecture of the work package G is therefore based on two levels:

- The monitoring tools and extraction tools which implement the concept of resource profile. These tools rely on measurement, target, history and filters, where filters provide a convenient way to extract well-adapted resource profiles. Main requirements of these tools are described hereafter (see section 2.2).
- A load-balancing policy based on the proposed monitoring tools and data-mining techniques. Since the design of ETEL++ is on going, we do not detail the corresponding load-balancing policy. This policy will be detailed in document DJ3 (Design of ETEL++). However, we provide some basic use-cases.

2.2 Monitoring Tools

Monitoring should typically reflect the utilization of CPU, disk, memory and network. The utilization of a resource such as a disk is described by means of a basic measurement reflecting for example the used storage space. These measurements are also attached to the notion of cluster (see

documents [DB3] and [DB4]), that is, the measurement may reflect the utilization of storage space of a cluster or a set of clusters. Therefore, we intend to provide the monitoring of hardware resources for various FollowMe targets like a host, a cluster or a set of clusters.

Applications also rely on more complex measurements. An application monitoring the bandwidth available on the network of a given host has to collect measurements of bandwidth available through various paths. These basic measurements can be compiled in a complex measurement reflecting more accurately the bandwidth of a given host. Work package G has therefore to provide a notion of aggregate of basic measurements.

In order to obtain trusted values, we also need to proceed with the compilation of successive values. These measurements are stored in a history. Dealing with the large number of values stored in a history is a difficult task based on the detection of representative values and tendencies. The computation of such meaningful values requires filtering of the history of measurements.

2.3 Load-Balancing Policy

In the context of FollowMe, we plan to use performance monitoring and algorithms that mine association rules to enhance the performance of the pilot application ETEL++. ETEL++ offers an online version of a regional newspaper. In addition to the traditional features of online services (like navigation, support of multimedia information, ...), ETEL++ will support personalization, i.e., ETEL++ will allow users to choose which areas the personalized newspaper will cover. The choices for a user are saved into his *user profile*.

Another feature of ETEL++ will be its ability to deliver data according to several different formats in order to support access from various types of terminals (e.g., PCs, portable phones, palmtops). Supporting many user-terminals (output devices) means that many different physical representations of the same logical data exist. All those representations find their origin in the same elementary structure (the real newspaper, directly out of the journalists hands), and differ in their media type. For example, an article represented originally as a text file can be converted into an audio file or a postscript file in order to be delivered respectively over a phone and on a laser printer. It is important to note here that the derived representations of a newspaper dramatically increase the volume of data to manage.

Load balancing policy has therefore to take into account the various constraints mentioned above:

- User’s mobility,
- Computation of several formats depending on user terminal,
- user preferences,
- The underlying architecture.

As described in section 5, the load-balancing policy has to map computations depending on trade-offs between communication costs and computation costs (see section 5.1). It has also to map users onto the architecture depending on their physical location and their preferences (see section 5.2).

3 A Short Survey

In this short survey, we focus on load-balancing techniques. These techniques mainly relies on the exploitation of profiles. The profiles referred as *User profile* may reflect both the user's preferences in terms of geographical location, used terminal and accessed services. W e also refer to *resource profile* which reflect the usage of the underlying system.

Monitoring tools allowing to compute profiles reflecting the actual state of the architecture. These tools provide various numbers, each reflecting the usage of a resource like the CPU, the memory, the disk or the network links. By providing resources' profiles, monitoring tools allow applications to take load-balancing related decisions. Section 3.1 details some problems making performance monitoring a complex issue. Section 3.2 describes several monitoring tools that are commonly used under Unix and Windows.

Exploiting user's profiles relies on data mining techniques. These techniques group users according to their preferences. Section 3.3 presents a short overview of these techniques, according to the needs of the deployment of ETEL++.

3.1 What to Monitor, How and When to Measure?

As soon as a computer is switched on, the operating system starts to monitor the resources it manages in order to deal with multiprogramming, sharing, memory allotment, scheduling, interrupts, etc... Resource monitoring is also possible for high-level applications because the operating system provides system calls that typically return elementary numbers reflecting the usage of a given resource. For example, Unix provides a handy call (`getrusage`) that returns information about resource utilization of the calling process (user and system times, numbers of input and output blocks, numbers of pages faults, ...).

The following resources are traditionally monitored: CPU load, disk utilization, memory consumption, I/O rate, caching hits, network packets loss, socket usage, etc...

On top of the basic calls provided by the operating system, tools or applications can build more sophisticated monitors to get indirectly numbers that reflect the utilization of a resource. A typical example is getting numbers reflecting the bandwidth of a network link. While no system calls directly deliver this information, the bandwidth can be for example deduced by observing the round trip time of a (say) 16K-bytes message. Note that the round trip time is obtained by invoking (at least twice) system calls returning the actual time of the day.

Regardless of getting performance directly or indirectly as mentioned above, monitoring tools are traditionally either exploiting immediately the numbers, or somehow compiling successive numbers in order to detect tendencies. The first approach is rather straightforward. The second approach raises the following issues.

In order to detect tendencies, monitoring tools have to get multiple and successive performance numbers. Deciding the appropriate frequency for sampling is rather difficult. Gathering numbers too frequently raises the risk of overloading the monitored resource with intrusive requests. A small sampling frequency is less disturbing, however, but raises the risk of missing major variations and thus the compilation of the gathered numbers can possibly be inaccurate. The frequency of sampling usually depends on the type of monitored resource, and is a trade-off between the additional load imposed over the resource and the confidence in the returned numbers. It also depends on the needs of the applications using the measurements. Typically, tools like `xload` or `taskmgr.exe` sample the system every second.

In addition to the frequency sampling issue, computing meaningful values based on the elementary numbers that are gathered is difficult. To isolate tendencies, to show the evolutions of resource usage, it is sometime necessary to filter-out transient bursts that may be irrelevant. This filtering is, as the frequency of sampling, a trade-off between being very reactive and returning numbers that show very short term phenomenon versus returning numbers that do not closely reflect the actual activity of the monitored resources but that may be more useful on the long term. Typically, filtering is based on the traditional notions of average, deviation, mean-square and smoothing. Another common type of filtering is based on a threshold: a number is returned only when the sample reflects an utilization that is above a given threshold. This typically requires an event manager or a notification mechanism.

Another issue -- of a higher level, however -- is the translation of the gathered numbers into a format that is useful for applications. Typically, tools transform the samples into arrays of numbers, histograms, graphs or pies.

3.2 A Short Overview of Existing Performance Monitoring Tools

This section presents several well-known tools that monitor systems, and focuses on tools that monitor the network. Their very goal is to deliver to humans numbers that reflect as closely as possible the activity of the monitored resources. None of these tools take decisions about what to do with the gathered data, that is, none try to balance the load of the system depending on the observed tendencies for example. Rather, these tools are “spies”, and they help users to take smart decisions about tuning their system by providing accurate, confident and fully trusted numbers. Although the focus of this section is network tools, we first, however, briefly mention typical tools that monitor machines.

3.2.1 Monitoring Machines

Every single machine provides low level system calls and some elementary tools of higher level that monitor the machine itself. Not surprisingly, monitoring is attached to the notion of process, that is, returned data shows how each process uses the resources. As the amount of information delivered for every process is rather large, it is sometimes better to merge process-based data and to extract

from this merging higher level notions like the resource usage of a group of process, or process that are owned by the same person, or process having parent-child relationships, etc…

The tools generally available deliver roughly the same information, and only their interface with the user makes each specific. For information, we could mention `xload`, `sar`, `vmstat`, `taskmgr` and `top` as the tools that are the most frequently available on Unix and Windows based systems.

3.2.2 Monitoring the Network

Since mid-1995, many groups initiated extensive monitoring of the best known wide area network: the Internet. Their goal was to encourage research laboratories and companies around the globe to initiate rigorous joint studies in order to (i) define good metrics to evaluate the performance of the Internet and (ii) increase the knowledge level we have about the behavior of the network by performing extensive measurements on the short and long term. Several working groups tackled this problem: The IP Performance Metrics and the Realtime Traffic Flow Measurement groups of the Internet Engineering Task Force (IETF) (see http://www.ietf.org), the Regional Internet Registry for Europe (RIPE) (see http://www.ripe.net), and others.

In general, these groups have performed extensive measurements, and some pieces of the software they developed is now publicly available. We mention here two such tools that can be reused on a smaller-scale basis.

The Realtime Traffic Flow Measurement group of IETF made publicly available their software called NeTraMet (see http://www.auckland.ac.nz/net/Internet/rtfm). NeTraMet is an accounting meter which runs on a PC under DOS or a Unix system. It builds up packet and byte counts for traffic flows, which are defined by their end-point addresses. Addresses can be Ethernet addresses, protocol addresses (IP, DECnet, EtherTalk, IPX or CLNS, IP port numbers, etc), or any combination of these. The traffic flows to be observed are specified by a set of rules. Traffic flow data is collected via SNMP from NeTraMet by a specific program. NeTraMet provides a valuable tool for analyzing network traffic flows, and should prove to be of interest to anyone interested in network monitoring, capacity planning, performance measurement, etc.

Another very powerful tool is NNStat. NNStat has been originally developed in 1992 by R. Braden and A. DeSchon at the University of Southern California, Information Sciences Institute. NNStat is a collection of programs that provides an Internet statistic collecting capability. The NNStat strategy for statistic collection is to collect traffic statistics via a promiscuous Ethernet tap on the local networks, versus instrumenting the gateways. If all traffic entering or leaving a network or set of networks traverses a local Ethernet, then by stationing a statistic gathering agent on each local network a profile of network traffic can be gathered. Statistical data is retrieved from the local agents by a global manager. The NNStat distribution comes with several sample `awk` programs which process the logged output of the collect program.

Most of the detailed softwares are based on the following tools:

- Tools similar to the “Ping” call of Unix
- The protocols SNMP and CMIP

SNMP has been designed in the mid-1980's as an answer to the communication problems between different types of networks. The way it works is very simple: it exchanges network information through messages (called protocol data units). From a high-level perspective, each message is an object that contains variables that have both names and values. There are five types of messages that

SNMP employs to monitor a network: two deal with reading terminal data, two deal with setting terminal data, and one, the trap, is used for monitoring network events such as terminal start-ups or shut-downs. Each variable consists of the following information:

- the name of the variable
- the data type of the variable (e.g., integer, string)
- whether the variable is read-only or read-write
- the value of the variable

Basically, SNMP allows to monitor each site running an SNMP agent. The main advantage of this protocol is that it is in wide use. Therefore, there are a lot of already available agents which provide monitoring of network activity and hardware characteristics. Some of them also monitor the usage of memory and disks.

3.3 Data Mining Techniques

The ever growing number of databases related applications (for the purpose of business, science and engineering for example) raised the need for new techniques and for new tools that can extract useful information from the stored data in order to provide valuable additional knowledge. This extraction, or discovery of knowledge in databases, is also called *data mining*. It corresponds to a non-trivial extraction of implicit, previously unknown and potentially useful information from data in databases [PSF91, ACF94, AMS96, Han96, FPSSU96]. By mining large databases, interesting knowledge, regularities, specific patterns and high-level information can be extracted and investigated from different angles. The discovered knowledge can be applied to information management, query processing, decision making, process control, artificial intelligence, statistics and data visualization. Furthermore, several emerging application for information providing services, such as on-line services and the World Wide Web, also call for various data mining techniques to better understand user behavior and to ameliorate the service provided. In general, data mining offers opportunities for major revenues.

Data mining poses many challenging issues. Due to space limitations, we unveil here only the tip of the iceberg: many other interesting issues are presented in [CHY96, CM96]. A first issue is related to the diversity of data types (typically ranging from relational data to complex objects) and to the different goals of mining. This diversity make it difficult for one mining system to handle all kinds of data. Rather, specific data mining systems are constructed, each offering dedicated solutions to knowledge mining. Another problem comes from the performance of mining algorithms. To effectively extract information from a huge amount of data, the mining algorithms must be efficient and scalable to very large data sets. This poses severe constraints on the complexity of the mining algorithms, and usually require elegant implementations. The usefulness, certainty, expressiveness and accuracy of the mined results is another source of hard problems. This becomes a major concern especially when the database used for mining is updated.

Data mining is an application-dependent issue, and different applications may require different mining techniques to cope with. In general, however, the kinds of knowledge which can be discovered fall in one of the following categories: mining association rules, multi-level data generalization, summarization and characterization, data classification, data clustering, pattern-based similarity search and mining sequential patterns. Each category is briefly detailed in the following sections.

3.3.1 Mining Association Rules

Typically, mining *association rules* means discovering in a database of sales transactions the important associations among items, such that the presence of some items in a transaction will imply the presence of other items in the same transaction. Two classical example are the discovery that ``80% of the customers purchasing milk also purchase bread" and that ``70% of the people buy wheat bread if they buy 2% milk". Such knowledge is useful for example to better organize the shelves, or to develop grouped promotions.

Mining association rules may require to repeatedly scan through the entire database to find different association patterns. As such, efficient algorithms and some methods for further performance enhancement are needed. [AIS93, AS94, PCY95] present typical algorithms that are efficient and also representative of mining association rules techniques.

Mining association rules can be expressed as follows. Let D be a set of transactions, such that each transaction contains a subset of items of the itemset I (e.g., I is the set of *all* the items you can purchase in your local store. A transaction is what you actually bought the other day). An association rule is an implication of the form $X \Rightarrow Y$, where $X \subset I$, $Y \subset I$ and $X \cap Y = \emptyset$. The rule $X \Rightarrow Y$ holds for the transaction set D with *confidence c* if c% of transactions in D that contain X also contain Y. The rule $X \Rightarrow Y$ has *support s* in the transaction set D if s% of transactions in D contains .

Confidence denotes the strength of implication and support indicates the frequency of the occurring patterns in rule. Only those rules that have a reasonably large support are considered.

The algorithms cited above typically implements the following two steps:

Discover the large itemsets L, i.e., the sets of itemsets that have a support above a pre-determined minimum support s.

For each large itemset $l \in L$, search for the association rules $l_1 \Rightarrow l_2$ with $l_1 \subset L$, $l_2 \subset L$, $l_1 \cap l_2 = \emptyset$ and $l_1 \cup l_2 = l$. Select the rules that have a confidence above a pre-determined minimum confidence c.

It is noted that the overall performance of mining association rules is determined by the first step -- deriving the rules from the itemsets being straightforward.

3.3.2 Multi-Level Data Generalization, Summarization and Characterization

Data often contain detailed information *at primitive concept levels*. It is often desirable to summarize a large set of data and present it at a high concept level. This functionality is achieved by the class of mining algorithms that are presented in this section. It is important to note that data generalization is the most popular way to use mining algorithms, often referred to as the *data cube approach* [Squ95, Moh 96, Inm 96, HRU 96, HAMS 97].

The general idea behind the data cube approach (also called OLAP, for On-Line Analytical Processing) is to materialize (i.e., pre-compute, typically off-line) and store certain *expensive* computations that are frequently inquired, especially those involving aggregate functions such as count, sum, average, max, etc. The materialized views are stored in a multi-dimensional database called a data cube (or a data warehouse) that can be interrogated for the purpose of knowledge discovery. Typically, values are grouped into a hierarchy. For example, the transactions reflecting the purchases of customers on a every day basis can be grouped into ``week", ``month" and ``year", and for each group, an aggregate value of the sales for that period is automatically computed. Generalizations

and specializations can be performed on the data cube by roll-up and drill-down operations. The data cube approach is a valuable technique for many business-oriented applications, since it provides many different views on the same basic data, as it enforces performances by pre-computing frequently asked questions. [CHY 96] also details a technique for on-line data generalization, called the attribute-oriented induction approach.

3.3.3 Data Classification

Data classification is the process which finds the common properties among a set of objects in a database and classifies them into different classes. The data is classified according to a model, which is usually constructed from a sample extracted from the real database. For example, it might be desirable for a insurance company to classify its customers according to the date of their last crash, and to subsequently mine that such customers are likely to be single, male, below 25 years old, living in large cities. A well know approach to data classification is the use of decision trees [Qui86, MRA95, SAM96]. In this approach, a small sample is first used to built an initial decision tree. If the tree does not model accurately enough the complete set of data, a selection of the exceptions is added to the sample, and the process continues until the correct decision tree is found.

3.3.4 Data Clustering

The process of grouping physical or abstract objects into classes of similar objects is called clustering. Clustering analysis helps construct meaningful partitions of a large set of objects based on a divide-and-conquer approach which decomposes a large scale system into smaller components. In the context of data mining, data clustering identifies densely populated regions, according to some distance measurement, in a large, multidimensional data set [NH94, ZRL96]. In other words, data clustering tries to discover the overall distribution patterns of a data set, and thus facilitates taxonomy.

3.3.5 Pattern-Based Similarity Search

Temporal or spatial-temporal data constitutes a significant portion of the data stored in databases. Typical examples are financial databases for stock price index and medical databases. Searching for similar patterns in such databases is useful to discover and predict the risk, causality and trend associated with a specific pattern. Classical examples of queries for this type of database include identifying companies with similar growth patterns, products with similar selling patterns, stocks with similar price movements, tumors with similar initial characteristics, similar weather patterns, etc.

4 Design

4.1 Overview

Monitoring tools of work package G rely on the notion of history. The history allows to collect values reflecting the performances of hardware resources such as hosts and network.

In the proposed design of work package G, the management of histories is addressed by a software component called Monitor. A Monitor is dedicated to a host and deals with the histories related to the performances of its hardware and software resources. A Monitor is responsible of creating histories and periodically collecting performance values in order to update them.

As soon as an History is created, applications can directly extract information without requesting the Monitor anymore. The extraction of information may be performed by means of Filters and Tests as described hereafter.

4.1.1 Class Diagram

To create an History, an agent has to request `Monitor` (Section 4.3 describes the actual interface). Each History is based on a `Measurement` which represents the resources the agent wants to monitor. As soon as the History is created, `Monitor` updates it by periodically collecting values of `Measurement.`

An application or an agent is able to create new `Measurement` reflecting its specific needs. `Measurement` may be a complex aggregate of various `basicMeasurement`. Each `basicMeasurement` reflects the way a `Target` uses a `Resource`:

- A `Resource` may be one of the hardware component of a host (Disk, CPU, Memory, Network…). A `Resource` may also be application-specific. For example, the DeviceGateway described in User Access [DH3] may be a monitored in terms of number of open connections.
- A `Target` may be either a cluster [DB3, DB4] or a set of clusters. We also provide a specific `Target` called `HOST` which is an exception reflecting the activity of all the objects of a given Host.

As an example, a `Measurement` may be built on the `Resource` CPU and a `Target` based on the Clusters of Etel++ (see section 5).

Figure 2 summarizes relationships between the main components we mentioned. Use of History by application is described more in details in section 4.2. In this diagram, application is considered as a client of Service Deployment package. This application may be either implemented by means of agents (see work package D) or mobile objects (see work package B). Some examples may be found in section 5.

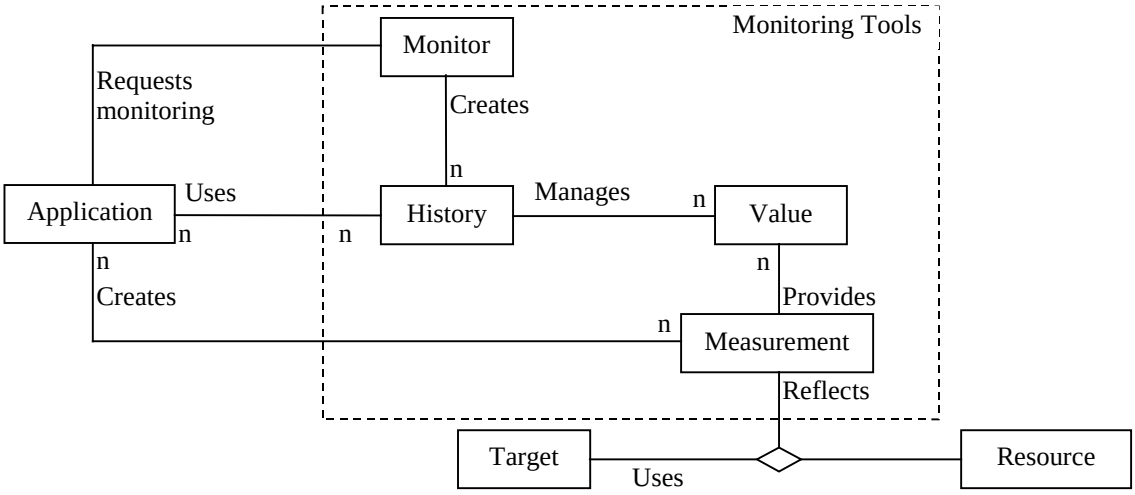

Figure 2 : Object model diagram

Details related to the concept of resource are provided in Figure 3. Note that monitoring hardware resources is implemented in the work package G whereas Application-specific resources can be implemented depending on needs of pilot applications.

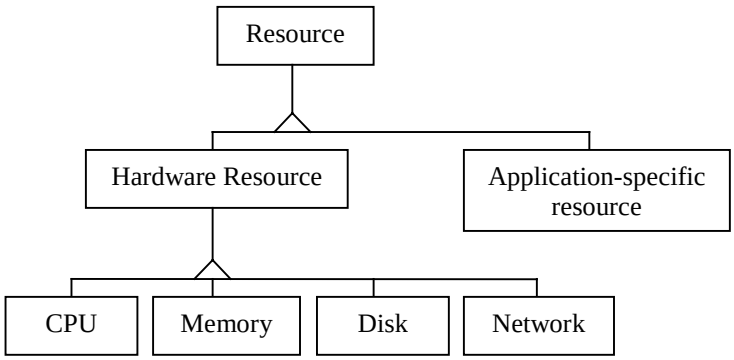

Figure 3 : Resources

4.2 Use of Histories

We provide a set of tools allowing the management of histories. As described in Figure 4, the application interrogates histories through various interfaces. The main goals of these interfaces are:

- To select the appropriate information in the large set of data gathered in Histories. This selection is performed by means of Filters. Filters can be used in a pipeline manner to exploit the data gathered in Histories. Section 4.2.1 provides an explanation of the notion of Filter and also some examples.
- To provide a well-adapted interrogation mode. The interrogation may be either based on request-mode or notification-mode (see section 4.2.2).

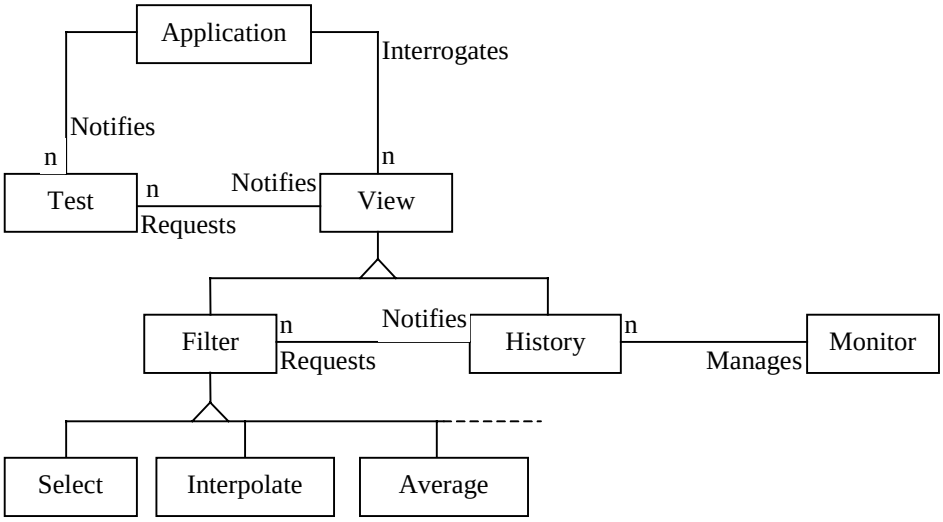

Figure 4 Use of Filters and Tests

4.2.1 Filters

The exploitation of measurements collected into Histories appears as a critical requirement in each monitoring tool. This exploitation relies on complex operations on a set of raw-values in order to compute higher level data more directly relevant to applications. In the proposed architecture, these operations are executed by mean of Filters.

Typical filters are for example:

- Average, minimal or maximal value over an interval of time, mean-square, etc…
- Linear interpolation, high- or low-gain filters…,
- Prediction of tendencies.

Filters can also be used in a pipeline manner for delivering sophisticated values based on several elementary filters. For example filters implementing selections and computing average values may be combined to eliminate transient bursts.

Work package G provides as a basis numerous filters listed in section 4.3.1. Application can also implement their own filters based on the abstract class Filter.

4.2.2 Request and Notification

There are two basic ways to use histories of measurements that have been compiled:

- Request-based mode. In this case, an application gets values upon explicit requests. A typical request needs the resource-name and a time range as arguments, and may return for example the average load placed on that resource during the specified period.
- Notification-based mode. In this case, an application gets values when an application-defined condition becomes true. To use an history in this mode, the application has to specify the name of the resource, a time range, and a condition. For example, the application can ask to be notified when the average load placed on that resource for the period becomes greater than a specific threshold value.

In general, the tradeoffs for choosing between these two modes are analogous to the ones that exist for the delivery of messages either upon request or via a notification service.

Filters and Histories both provide a Request-based mode. This mode is described in a common interface called View. As detailed hereafter (see section 4.3.4), view provides a set of methods allowing to evaluate the value stored in an History in a given time range.

The request-based mode used between an application and the various components of work package G is depicted in Figure 5. This example represents the interrogation of an History through a Filter.

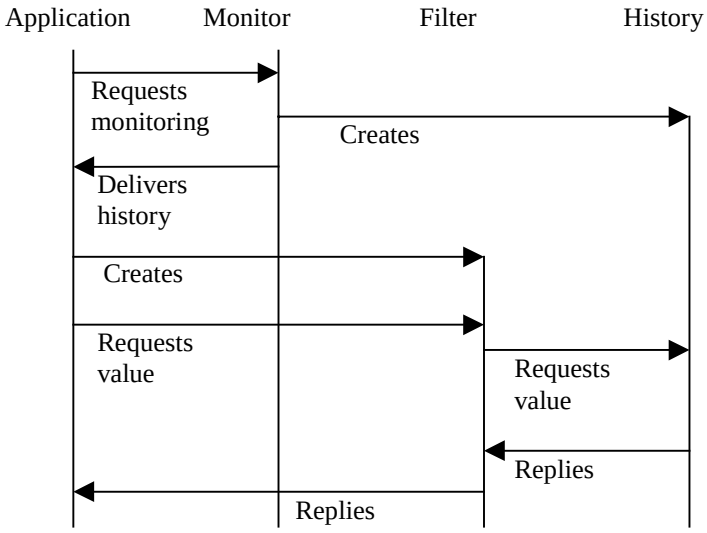

Figure 5 : Diagram of Request-based mode

This diagram corresponds to the code presented in Figure 6 Note that an overview of the interface used in this example can also be found Section 4.3.

```
History h;
Measurement m = new basicMeasurement(CPU(), new Target(HOST()));
/*
 * creation of a basicMeasurement for monitoring
 * the utilization of the CPU in a given host
 */
h = Monitor.monitor(m, new Frequency(20000));
/*
```

```
 * creation of the History corresponding to the basicMeasurement
 */
Filter f = Interpolation(h);
Value v = f.getEvaluation(date);
/*
 * the exploitation of this History is made on a Request-mode
 * through an Interpolation Filter
 */
```

Figure 6 : Use of Request-based mode

The notion of Test allows a service to be notified when an application-defined condition becomes true. Creation of Test and its use are described Figure 7.

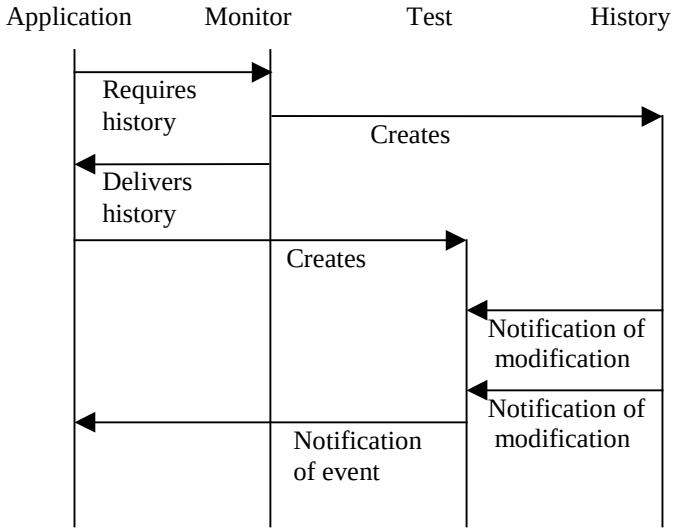

Figure 7 : Diagram of Notification-based mode

Notice that both Histories and Filters may be used on a Notification-based mode through a Test. This diagram corresponds to the code written in Figure 8.

```
History h;
Measurement m = new basicMeasurement(CPU(), new Target(HOST()));
h = Monitor.monitor(m, new Frequency(20000));

Test t = new myTest(h);
/*
 * creation of myTest which is an application-specific Test
 * plugged on the output of History h.
 */

Observer o = new myObserver ();
t.addObserver(o);
```

```
/*
 * Test is Observable. The application may execute the adequate
 * processing by implementing an Observer.
 */
```

Figure 8 : Use of Notification-based mode

4.3 API

We present the main classes of the proposed API of work package G. We do not describe the exhaustive API which can be retrieved on http://hyperwav.fast.de/inria/WPGinterface/.

4.3.1 Class Hierarchy

- class java.lang.Object
  - class ServiceDeployment.Monitor
  - class ServiceDeployment.Frequency
  - interface ServiceDeployment.Measurement
  - class ServiceDeployment.MeasurementAggregate (implements ServiceDeployment.Measurement)
  - class ServiceDeployment.basicMeasurement (implements ServiceDeployment.Measurement)
  - interface ServiceDeployment.Value
  - class ServiceDeployment.ValueAggregate (implements ServiceDeployment.Value)
  - class java.util.Observable
    - class ServiceDeployment.View
      - class ServiceDeployment.History
      - class ServiceDeployment.Filter (implements java.util.Observer)
        - class ServiceDeployment.Interpolation
        - class ServiceDeployment.Select
        - class ServiceDeployment.Average
    - class ServiceDeployment.Test (implements java.util.Observer)
  - class ServiceDeployment.Resource
    - class ServiceDeployment.CPU
    - class ServiceDeployment.Disk
    - class ServiceDeployment.Memory
    - class ServiceDeployment.Network
  - class ServiceDeployment.Target
  - class java.lang.Throwable (implements java.io.Serializable)
    - class java.lang.Exception
      - class ServiceDeployment.NoEvaluationException

4.3.2 Class ServiceDeployment.Monitor

```
java.lang.Object
   |
   +----ServiceDeployment.Monitor
```

public class **Monitor**

extends Object

Monitor Manages the Histories. It requests values of Measurements periodically and delivers them to Histories. Services may also request a single value to Monitor by mean of Monitor.getEvaluation(Measurement m).

Version:

March 1998

Author:

mb

See Also:

Measurement, History

```
public Monitor ()
```

Constructor of Monitor. There should be only one Monitor on a Host.

```
public History monitor (Measurement m, Frequency f)
```

Method monitor allows an application to get an History and to start the updating of the History with the provided frequency.

```
public void stopMonitoring (History h)
```

Method stopMonitoring allows an application to stop the updating of the History. This method does not destroy the History.

4.3.3 Class ServiceDeployment.basicMeasurement

```
java.lang.Object
   |
   +----ServiceDeployment.Measurement
           |
           +----ServiceDeployment.basicMeasurement
```

Version:

March 1998

Author:

mb

public class **basicMeasurement**

extends Measurement

```
public basicMeasurement (Resource r, Target t)
```

Constructor allows the application to define the Resource and Target to monitor.

```
protected Value getValue ()
```

Applications do not access this method which is the way Monitor updates an History.

Overrides:

getValue in class Measurement

```
public Resource getResource ()
```

Delivers the Resource specified in Measurement.

```
public Target getTarget ()
```

Delivers the Target specified in Measurement.

4.3.4 Class ServiceDeployment.View

```
java.lang.Object
   |
   +----java.util.Observable
           |
           +----ServiceDeployment.View
```

public abstract class **View**

extends Observable

superclass of History and Filter. View defines the interface applications can access on a Request-based mode. View provides a set of methods allowing to evaluate the value stored in an History in a given time range.

Version:

March 1998

Author:

mb

See Also:

History, Filter

```
public abstract Value getEvaluation (Date date) throws NoEvaluationException
```

Delivers the Value corresponding to a given Date. Type of Date is defined in Java.util.Date.

```
public abstract Date getBegin () throws NoEvaluationException
```

Delivers the Date corresponding to the first Value.

```
public abstract Date getEnd () throws NoEvaluationException
```

Delivers the Date corresponding to the last Value.

```
public abstract boolean isEmpty ()
```

Delivers true if View does not store any Value.

```
public abstract Value getValueAfter (Date date) throws NoEvaluationException
```

Delivers the first Value of View after Date.

```
public abstract Value getValueBefore (Date date) throws NoEvaluationException
```

Delivers the first Value of View before Date.

```
public abstract Date getDateAfter (Date date) throws NoEvaluationException
```

Delivers the first Date of View after Date.

```
public abstract Date getDateBefore (Date date) throws NoEvaluationException
```

Delivers the first Date of View before Date.

```
public abstract String getType ()
```

4.3.5 Class ServiceDeployment.Test

```
java.lang.Object
   |
   +----java.util.Observable
           |
           +----ServiceDeployment.Test
```

public abstract class **Test**

extends Observable

implements Observer

Test is applied on the output of History or Filter in order to provide notification-based interface.

Version:

March 1998

Author:

mb

See Also:

Measurement, History

```
public Test (View v)
```

Constructor of Test. V is the View Test has to be plugged on.

```
public abstract boolean evaluate ()
```

implements the test to apply on View. If evaluate delivers true the service is notified.

4.3.6 Class ServiceDeployment.History

```
java.lang.Object
   |
   +----java.util.Observable
           |
           +----ServiceDeployment.View
                   |
                   +----ServiceDeployment.History
```

public class **History**

extends View

History gathers values delivered by Measurement. Each value is associated to the corresponding Date. History provides the interface of View in order to retrieve a value before or after a date. Filters and Tests may also be applied on the output of History in order to provide adequate output. There is no publis interface to History since History is delivered and updated by Monitor. The only way to interrogate History is specified in the View interface.

Version:

March 1998

Author:

mb

See Also:

Filter, Test, Value, Monitor

5 Use-Cases

This section aims to present examples of load-balancing policies. These examples are expected to outline some useful features of FollowMe. Section 5.1 presents the use of code mobility in order to balance the computation and communication costs. Then, section 5.2 presents the exploitation of users' profiles to group users and to reduce the load of servers.

5.1 Exploiting Performance Monitoring to Balance Computation and Communication Costs

From a bird's eye-view, the architecture on top of which ETEL++ is built is made of three entities:

1) A central server located at Ouest-France (information provider). On this server are stored the articles once they are produced by the journalists. This server is connected to several secondary servers described next.

2) Secondary servers that are FollowMe-enabled machines. Each of these server has storage and computing power. To a secondary server are connected a set of user terminals described next.

3) User terminals that can be for example personal computers. A user ultimately browses its personalized electronic version of the newspaper on his terminal.

Allowing a user to browse his newspaper essentially consists in (i) determining what type of terminal is used, (ii) what are the chosen areas of information to cover and (iii) sending the relevant data from the central server to the user terminal. It is important to note here that the original (raw) data is created inside the Ouest-France server and that the derived data is displayed on users' terminal. As such, it is crucial to determine where that transformation takes place. Not surprisingly, three locations have to be considered:

- Inside Ouest-France's central server. In this case, all the possible versions are computed once, and the relevant versions can be either pushed or pulled by each secondary server before being delivered to users. The load on these secondary servers or on the terminal used by the users is almost zero: all the computations have been already done. The network, however, could become a bottleneck and might be severely congested. In addition, note that the Ouest-France machine is also heavily loaded.

- Inside each secondary server. With this approach, raw data is transmitted to these servers. Raw data tend to be of a much smaller size than the derive counterpart, and only a single version of that raw data travels around. In this case, all secondary servers have to compute their own derived versions. Two different servers will certainly make redundant computations since they are likely to generate at least a certain number of similar versions. With this approach, the burden is better distributed. Secondary servers, however, need a fairly reasonable amount of storage to keep the derived versions, and must also have a good processing power. This could not be the case for every single machine, consequently creating another set of problems.
- Inside the user's terminal. In this case, only the relevant version is computed using the raw data transmitted to the user terminal itself. In this case, the user terminal is loaded, and must be able to perform this computation efficiently. This approach might not be practical since user terminals are likely to have very different power and storage characteristics. Furthermore, storage space and computing power is somehow wasted since secondary servers are not involved.

It is possible to trade the costs of computing the derived versions against the costs of sending the versions via the network by appropriately choosing the location(s) of data processing. Computing the derived data very early (i.e., close to the central server) reduces computing costs (done once) while it increases the communication costs. The opposite applies if the derived data is computed very late, that is, close to the users. A flexible computing strategy would be to compute the derived versions using a carefully chosen combination of the previous locations. For example, the central server may compute a subset of the derived data, the rest may be divided among the secondary servers, and communications between servers could be enforced to leverage. In this case, the degree of computation redundancy, the amount of replication could be set such that it minimizes the overall cost.

Determining the best configuration is hard, and is certainly a NP-complete problem. Monitoring tools might be helpful in this case, because they can be used to define heuristics improving the behavior of the system. Observing the load of resources, and also being able to foresee a possible behavior of those resources might help ETEL++ in finding a good placement of its computations and its data in order to maximize its efficiency.

5.2 Mining User Profile to Group Users on Servers

To enforce the quality of service offered by ETEL++, we try to build a load-balancing policy that is partly based on the mining of association rules between the profiles of users. By grouping users that have close profile in terms of their personal choices, it is possible to assign group of users to specific servers that would best serve their requests. Assigning groups to the relevant servers help to balance the load across all the available servers, each managing a reduced set of users.

We specifically intend to use mining algorithms to find the themes users are likely to access frequently (i.e., large itemsets using the data mining vocabulary). Given that knowledge, groups can be created and then assigned onto servers. While the use of mining technology is rather straightforward in our case, finding the best mapping in order to create groups that will improve significantly the performance is still an issue. Finding efficient solutions is part of our ongoing work.

6 Status

The main goals of work package G are as follows:

- Providing monitoring tools. These tools are mainly based on Histories and Filters.
- Implementing a specific load-balancing policy dedicated to Etel++. This policy is based on both monitoring tools an data mining techniques.

Three documents have been produced (Survey, Requirements and Design). The design of the load-balancing policy relies on the design of Etel++ which is in preparation. Therefore, this document focuses on monitoring tools.

The status of each part is as follows:

- Monitoring tools are designed. The implementation is well advanced. However, much work is needed on the implementation of specific resources such as Network.
- The design of a load-balancing policy fulfilling the requirements of Etel++ has been initiated.

7 References

DJ2

INRIA, TCM. Requirements of Pilot Application 2: Etel++ (Work Package J). March 1998.

DB3

APM. Design of Mobile Object Workbench (Work Package B). December 1997.

DB4

APM. Interface of Mobile Object Workbench (Work Package B). December 1997.

DH3

FAST. User Access (Work Package H). January 1998.

ACF

Rakesh Agrawal, Michael J. Carey, Christos Faloutsos, Sakti P. Ghosh, Maurice A. W. Houtsma, Tomasz Imielinski, Balakrishna R. Iyer, A. Mahboob, H. Miranda, Ramakrishnan Srikant, and Arun N. Swami. Quest: A project on database mining. In Richard T. Snodgrass and Marianne Winslett, editors, *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, page 514, Minneapolis, Minnesota, 24-27 May 1994.

AFS93

Rakesh Agrawal, Christos Faloutsos, and Arun Swami. Efficient similarity search in sequence databases. *Lecture Notes in Computer Science*, 730:69, 1993.

AIS93

Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In Peter Buneman and Sushil Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207-216, Washington, D.C., 26-28 May 1993.

ALSS95

R. Agrawal, K.-I. Lin, H. S. Sawhney, and K. Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In Umeshwar Dayal, Peter M. D. Gray, and Shojiro Nishio, editors, *VLDB '95: proceedings of the 21st International Conference on Very Large Data Bases, Zurich, Switzerland, Sept. 11-15, 1995*, pages 490-501, Los Altos, CA 94022, USA, 1995. Morgan Kaufmann Publishers.

AMS96

Rakesh Agrawal, Manish Mehta, John Shafer, Ramakrishnan Srikant, Andreas Arning, and Toni Bollinger. The quest data mining system. page 244. AAAI Press, 1996.

AS94

R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In Jorgeesh Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago, Chile proceedings*, pages 487-499, Los Altos, CA 94022, USA, 1994. Morgan Kaufmann Publishers.

CHY96

Ming-Syan Chen, Jiawei Han, and Philip S. Yu. Data mining: an overview from a database perspective. *Ieee Trans. On Knowledge And Data Engineering*, 8:866-883, December 1996.

CM96

Chris Clifton and Don Marks. Security and privacy implications of data mining. In *Workshop on Data Mining and Knowledge Discovery*, number 96-08, pages 15-19, Montreal, Canada, June 2 1996. ACM SIGMOD, University of British Columbia Department of Computer Science.

CPY96

M.-S. Chen, J. S. Park, and P. S. Yu. Data mining for path traversal patterns in a web environment. In *ICDCS '96; Proceedings of the 16th International Conference on Distributed Computing Systems; May 27-30, 1996, Hong Kong*, pages 385-393, Washington - Brussels - Tokyo, May 1996. IEEE.

FPSSU96

U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. MII Press, Mento Park, 1996.

FRM94

C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 23(2):419-429, June 1994.

HAMS97

Ching-Tien Ho, Rakesh Agrawal, Nimrod Megiddo, and Ramakrishnan Srikant. Range queries in OLAP data cubes. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 26(2):73, 1997.

Han96

Jiawei Han. Data mining techniques. In H. V. Jagadish and Inderpal Singh Mumick, editors, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, page 545, Montreal, Quebec, Canada, 4-6 June 1996.

HRU96

Venky Harinarayan, Anand Rajaraman, and Jeffrey D. Ullman. Implementing data cubes efficiently. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 25(2):205, 1996.

Inm96

W. H. Inmon. The data warehouse and data mining. *Communications of the ACM*, 39(11):49-50, No-

vember 1996.

Moh96

N. Mohan. DWMS: Data warehouse management system. In T. M. Vijayaraman et al., editors, *Proceedings of the twenty-second international Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 588, Los Altos, CA 94022, USA, 1996. Morgan Kaufmann Publishers.

MRA95

Manish Mehta, Jorma Rissanen, and Rakesh Agrawal. MDL-based decision tree pruning. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD'95)*, pages 216-221, August 1995.

CY95

Jong Soo Park, Ming-Syan Chen, and Philip S. Yu. An effective hash based algorithm for mining association rules. In Michael J. Carey and Donovan A. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 175-186, San Jose, California, 22-25 May 1995.

PSF91

Gregory Piatetsky-Shapiro and William Frawley, editors. *Knowledge Discovery in Databases*. The MIT Press, Cambridge, MA, 1991.

Qui86

J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1:81-106, 1986.

SAM96

J. Shafer, R. Agrawal, and M. Mehta. SPRINT: A scalable parallel classifier for data mining. In T. M. Vijayaraman et al., editors, *Proceedings of the twenty-second international Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 544-555, Los Altos, CA 94022, USA, 1996. Morgan Kaufmann Publishers.

Squ95

Cass Squire. Data extraction and transformation for the data warehouse. In Michael J. Carey and Donovan A. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 446-447, San Jose, California, 22-25 May 1995.

ZRL96

Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: An efficient data clustering method for very large databases. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 25(2):103, 1996.