An infrastructure for the implementation of dynamic data warehouses

Vaishnavi Sailaja Madineni

Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd

Recommended Citation
https://lib.dr.iastate.edu/rtd/19493

This Thesis is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
An infrastructure for the implementation of dynamic data warehouses

by

Vaishnavi Sailaja Madineni

A thesis submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Leslie Miller, Major Professor
Johnny S.K. Wong
Doug Jacobson

Iowa State University
Ames, Iowa
2003

Copyright © Vaishnavi Sailaja Madineni, 2003. All rights reserved.
Graduate College
Iowa State University

This is to certify that the master’s thesis of

Vaishnavi Sailaja Madineni

has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy
ACKNOWLEDGEMENTS

I would like to thank my major professor Dr. Leslie Miller for his vision, constant guidance, support and assistance through out this project.

Also among ISU faculty I would thank Dr. Johnny Wong and Dr. Doug Jacobson for taking the time to serve on my Program of Study committee.

Among fellow students I am indebted to Nikhil Sathe and Sheng Qu for all the help they have provided in setting up the systems.

Finally, I am thankful to my family and friends for their unconditional support and encouragement, which has made everything worthwhile.
# TABLE OF CONTENTS

1. ABSTRACT ................................................................................................................................. vi

2. INTRODUCTION .......................................................................................................................... 1

3. BACKGROUND............................................................................................................................. 4
   2.1 Intrusion Detection ................................................................................................................ 4
   2.2 Data Warehouse ..................................................................................................................... 6
   2.3 Mobile Agents ......................................................................................................................... 9

4. TAXONOMY AND BASIC DESIGN OF A DATA WAREHOUSE ............................................. 13
   3.1 Taxonomy Tuple ................................................................................................................... 13

5. DYNAMIC DATA WAREHOUSE MODEL .............................................................................. 20
   4.1 Dynamic Data Warehouse Design ....................................................................................... 20

6. IMPLEMENTATION ISSUES ...................................................................................................... 23
   5.1 Dynamic Data Warehouse Prototype .................................................................................. 23
   5.2 Detailed Discussion of the System ...................................................................................... 24

7. EVALUATION .............................................................................................................................. 33
   6.1 Application Area Examples ................................................................................................. 33
   6.2 Issues in the Dynamic Data Warehouse ............................................................................. 34

8. INTRUSION DETECTION SYSTEM MODEL ............................................................................ 37
   7.1 Mobile Agent Intrusion Detection System: MAIDS ............................................................ 37
   7.2 MAIDS with a Data Warehouse ......................................................................................... 39

9. CONCLUSION .............................................................................................................................. 44

REFERENCES ..................................................................................................................................... 45

APPENDIX ......................................................................................................................................... 49
LIST OF FIGURES

3.1 Building Blocks of a Data Warehouse .................................................... 14
3.2 Different Approaches to Represent Data to Check for Intrusions .................... 17
4.1 Design of a Dynamic Data Warehouse .......................................................... 21
5.1 System Architecture of Dynamic Data Warehouse Prototype .................................. 23
5.2-5.10 Implementation Screenshots ................................................................. 25
7.1. Architecture of Mobile Agent Intrusion Detection System ................................. 38
7.2. IDS Shown in Detail with Wrappers ............................................................... 40
7.3. Working of Dynamic Data Warehouse ........................................................... 42
ABSTRACT

Data Warehouses have become a critical part of many organizations knowledge management system. The typical implementation if a data warehouse is designed to support basic data types usually found in legacy and relational database systems. Data Warehousing has always been associated with a large storage capacity data source that assists management of an organization in making strategic decisions. The main misconceptions regarding data warehousing are that it is a task of enormous effort and that it is useful for only large organizations having huge amount of transaction data. In addition data warehouses are usually static. While these systems are extremely valuable there are applications where the static nature of these warehouses is a hindrance. To more completely support such applications a model of the dynamic data warehouse is presented. This warehouse is built, operated and then brought down using mobile agents.
1. INTRODUCTION

The amount of data available to organizations today is staggering. In a typical organization there is a wide range of analysis needs, as well, depending upon the kind of applications. While some applications use the data to make instant decisions, some make use of this data to store knowledge in the form of rules for decisions to be made in near future. Thus while the data can be outdated for some applications, the same outdated data might be necessary for some other application. While current data warehouses have become an important part of any organization's knowledge management system, they have a fairly narrow focus relative to the needs of most organizations. Typically, they support efforts to analyze traditional value based data found in legacy systems and relational database systems. Current data warehouses are typically large static systems that are visible to a variety of users in the organization. Statistical analysis and data mining tools are usually available in such warehouses.

Data Warehouses have always been assumed to be bulky and static, something that occupies a large amount of resources and hence is too bulky to be moved around and used in a mobile environment. This assumption however is incorrect [24]. Data Warehousing is an activity that is independent of the size of data used and can be performed at any level of data size. Also, data warehousing is not something that necessarily requires a large amount of resources and time to develop. In most cases of current use that is true but it is not a necessity. Data warehouses are typically assumed to be static in nature. Also, the current warehouse architecture is decided upon in the design phase and remains the same throughout its life. The only part that changes is the data being added or deleted from the transactional data.

There are many non-traditional applications that are not able to take full advantage of the existing warehouses. These applications typically need data type support and tool types that are generally not available in commercial data warehouses. There have been several past efforts, which talk about the development of static data warehouse that supports more complex data types [2][23][15].
While on one hand there are applications that need more complex data types, the other hand there are applications that require the concept of a data warehouse with a less static structure. Two needs that showed up during this analysis are, first the need to bring the analysis power to the data when the volume of the data is very high and the size of the results of the analysis is relatively small. A second issue comes up in environments like security systems. A data warehouse is an attractive part of an intrusion detection system [18], but a static data warehouse creates a new target for hackers. To deal with such applications, a model of a dynamic data warehouse is developed that is capable of moving computing or analysis power to any computer in the network.

This thesis document is organized into 7 chapters. Chapter 2 gives background information about data warehouses, mobile agents that are used for the implementation of dynamism in the warehouses, and the advantages of mobile agents. It also talks briefly about intrusion detection in general. Chapter 3 looks at the taxonomy of a general warehouse model. A warehouse is defined to be composed of seven different components and each of these components is briefly described in terms of their functionality and necessity.

Chapter 4 presents a model of the dynamic data warehouse derived from the model presented in the Chapter 3. It is thus the selection of different options for each of the parameter in the general model of the warehouse with the possible elimination of some of the optional parameters as is described in Chapter3. Chapter 5 presents a prototype of dynamic data warehouse including its implementation details and other possible configurations. It gives a detailed description of the different classes being used for the implementation of the warehouse. Also through the use of screen shots the flow of control while using the warehouse is explicitly shown. Chapter 6 talks about the different applications for which the dynamic data warehouse are useful. Also, the chapter discusses some of the issues raised during the development of the working prototype. Chapter 7 discusses the Mobile Agent Intrusion Detection System and talks about the model in which a static/dynamic data warehouse replaces the database in the Intrusion detection system. Chapter 8 provides concluding remarks with comments on the future work to be done in the area of dynamic data warehouses.
The document presents a dynamic warehouse model which is not very time consuming to deploy, lightweight enough such that it can be built, operated and destroyed when required. This lightweight warehouse has the ability to move from location to location. Due to all these features this lightweight data warehouse is truly dynamic.
This chapter begins with a brief overview of computer security, misuse and anomaly detection. Then the concept of data warehouse and the general issues of a data warehouse are introduced to form a basis for later chapters that discuss key components of a general warehouse. Then the concept of mobile agents their advantages in comparison to the static agents are introduced.

2.1 Intrusion Detection

Intrusion detection is needed in today’s computing environment because it is impossible to keep pace with the current and potential threats and vulnerabilities in our computing systems. The environment is constantly evolving and changing fueled by new technology and the Internet. To make matters worse, threats and vulnerabilities in this environment are also constantly evolving. Intrusion detection products are tools to assist in managing threats and vulnerabilities in this changing environment.

Threats are people or groups who have the potential to compromise the computer system. Vulnerabilities are weaknesses in the systems. Vulnerabilities can be exploited and used to compromise the system [4][6].

There are two words to describe the intruder: hacker and cracker [4]. A hacker is a generic term for a person who likes getting into things. The benign hacker is the person who likes to get into his/her own computer and understand how it works. A benign hacker is thus referred to as a cracker.

2.1.1 The primary ways an intruder can get into a system [4][20]:

i. **Physical Intrusion** If the intruder has physical access to a machine (i.e. they can use the keyboard or take apart the system), he will be able to get in. Techniques range from special privileges the console has, to the ability to physically take apart the
system and remove the disk drive (and read/write it on another machine). Even BIOS protection is easy to bypass: virtually all BIOS's have backdoor passwords.

ii. **System Intrusion** This type of hacking assumes the intruder already has a low-privilege user account on the system. If the system doesn't have the latest security patches, there is a good chance the intruder will be able to use a known vulnerability in order to gain additional administrative privileges.

iii. **Remote Intrusion** This type of hacking involves an intruder who attempts to penetrate a system remotely across the network. The intruder begins with no special privileges. There are several forms of this hacking. For example, an intruder has a much more difficult time if there is a firewall between him/her and the victim machine.

An intruder can get into the system because of software bugs by exploiting the buffer overflows, race conditions, unhandled input (robustness), password cracking, sniffing unsecured traffic, and exploiting design flaws.

### 2.1.2 How are intrusions detected? [4][20][6]

i. **Anomaly Detection**

The most common way people approach network intrusion detection is to detect statistical anomalies. The idea behind this approach is to measure a "baseline" of such statistics as CPU utilization, disk activity, user logins, file activity, and so forth. Then, the system can trigger when there is a deviation from this baseline. The benefit of this approach is that it can detect the anomalies without having to understand the underlying cause behind the anomalies. Also it may not be necessary to have examples of misbehavior to develop a profile of expected or predicted behavior. An anomaly detection system also does not depend on the prior knowledge of possible misuses and can help discover new patterns of misuse. The disadvantage of such systems is that they generally suffer from a large number of false detections as underlying use of the network shifts.

ii. **Misuse Detection**
A misuse intrusion detection system searches for occurrences of events, which have previously been identified as intrusions. Misuse intrusion detection systems detect matching intrusions with reasonably good accuracy. Misuse based intrusion detection systems have several disadvantages. Only known intrusions can be identified, so newly discovered vulnerabilities can be exploited without being detected. Also signatures may be difficult to develop for certain intrusions.

2.2 Data warehouse
A Data Warehouse is a repository of integrated information and tools, available for queries and analysis [1][13]. Data and information are extracted from heterogeneous sources as they are generated. This makes it much easier and more efficient to run queries over data that originally came from different sources. Data warehouses have become an instant phenomenon in many large institutions that deal with a massive amount of information. The first wave of warehouse systems had been primarily aimed at processing typical data found in relational databases and legacy file systems. As the value of data warehouses has become more apparent to companies, users have been looking at incorporating other types of data into the warehouses making use of the object-oriented databases. A data warehouse is not a standalone system. It only serves as a central repository for some portion of an organization's memory. Thus we use a local data warehouse to refer to the portion of the data warehouse that stores data, stores and initiates tools and queries the data. Thus the data warehouse should be able to integrate the data sources of the organization's memory and extract the portion of the data that is required to support the decision making process.

2.2.1 Data Warehouse Issues [15][2]
According to W.H.Inmon[1] data warehouse is a "subject oriented, integrated, time-varying, non volatile collection of data that is used primarily in organizational decision making. There are a set of standard data warehouse issues which separate it from a normal traditional database and which need to be addressed properly in order to understand the importance.
Table 2.1 Comparisons between a Data Warehouse and an Operational Database

<table>
<thead>
<tr>
<th>Data Warehouse</th>
<th>Operational Database</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subject oriented</strong></td>
<td><strong>Application oriented</strong></td>
</tr>
<tr>
<td>Data Warehouses are built around broad non-overlapping subjects rather than individual transactions. Thus data congregate around categories rather than individual transactions.</td>
<td>An Operational Database is built around the kind of application the database is used for.</td>
</tr>
<tr>
<td><strong>Integrated</strong></td>
<td><strong>Limited integration</strong></td>
</tr>
<tr>
<td>The data warehouse consolidates application-oriented data from different legacy systems, which involve various encodings, measurement units, etc.</td>
<td>An Operational database is used for storing data of a particular format. It offers only limited integration through the use of wrappers</td>
</tr>
<tr>
<td><strong>Nonvolatile</strong></td>
<td><strong>Continuously updated</strong></td>
</tr>
<tr>
<td>New data is always appended to the database, rather than replacing it, integrating it with the previous data.</td>
<td>The database at any instant represents the current state and since the calculations are based on the current state of data hence continuously updated.</td>
</tr>
<tr>
<td><strong>Stabilized data values</strong></td>
<td><strong>Current data values only</strong></td>
</tr>
<tr>
<td>Since a data warehouse essentially used for generating standardized rules, we need to have data which is “good” in the sense stabilized enough to produce good rules.</td>
<td>An Operational database need to have the current state of database for performing operations and has only the current values.</td>
</tr>
<tr>
<td><strong>Ad hoc retrieval</strong></td>
<td><strong>Predictable retrieval</strong></td>
</tr>
<tr>
<td>The user can present queries at random</td>
<td>more predictable pattern is preferred rather than random sequencing</td>
</tr>
</tbody>
</table>
We briefly overview each topic:

i. **Subject Orientation**: In a data warehouse, data is organized around major subjects of the enterprise rather than individual transactions. Data warehouse data thus is designed to aid in decision making instead of being designed to support application processing.

ii. **Data Integration**: the most important aspect of the data warehouse environment is that data found within the data warehouse is integrated. The integration shows up in many different ways - in consistent naming conventions, inconsistent measurement of variables, in consistent encoding structures, in consistent physical attributes of data, and so forth. Thus in a data warehouse the information should be clean, validated and properly aggregated. By clean we mean that the same piece of information must be referred to as represented uniformly. When data is brought into a warehouse, it is integrated and stored in a single globally acceptable fashion regardless of different data sources.

iii. **Non-volatile Environment**: updates - inserts, deletes, and changes - are done regularly to the operational environment on a record-by-record basis. But the basic manipulation of data that occurs in the data warehouse is much simpler. There are only two kinds of operations that occur in the data warehouse - the initial loading of data, and the access of data. There is no update of data (in the general sense of update)[14] in the data warehouse as a normal part of processing. Thus data in data warehouses are not updated regularly on a record-by-record basis. Rather data is integrated into the warehouse in a scheduled basis.

iv. **Time series**: Typically the data is accurate when the data is being loaded into the warehouse, while in an operational environment, decisions are made online therefore data must be accurate at moment of access. The data warehouse typically has a 5-10 year horizon on the data that it contains. Operational data on the other hand, typically covers 60 to 90 days.
2.2.2 Data Warehouse Topologies

Data Warehouse components can be configured using different topologies [13][15]:

i. **Centralized Data Warehouse.** A data warehouse contains properly conditioned data on subjects of interest to users within an enterprise’s multiple business units. A data warehouse supports cross-functional information requirements. A centralized data warehouse can be topologically simple, because it is the sole locus of warehouse data for many users (or clients) and applications throughout the organization. It is the topology of choice for many large data warehouses seeking the advantages of economy of scale and centralized system management.

ii. **Data Warehouses and Data Marts.** A data warehouse is often contrasted with a data mart, which typically contains a narrower scope of data, characterized by a single subject, a single business function, or even a single application. Data marts are often connected to a centralized warehouse in a Three-tier configuration in which clients are connected to specific data marts that draw their data from a data warehouse. This topology exploits locality of reference to provide optimal performance to the warehouse’s data-mart clients while allowing user access to warehouse data in order to meet cross-functional information requirements.

iii. **Hybrid deployment strategy:** Use of data marts is often restricted so that to avoid proliferation of independent data marts modeled and designed with out regard for cross-functional information requirements. Thus for many organizations, business objectives and resource constraints favor the faster bottom up data mart deployment strategy combined with a top down high level data model. Such a hybrid deployment begins with development of one or more data marts on a staged basis. Capabilities for cross functional processing are anticipated during the planning phase and are implemented incrementally as the system grows.

2.3 Mobile Agents

A software agent can be defined as [5]:
"A software entity which functions continuously and autonomously in a particular environment ... able to carry out activities in a flexible and intelligent manner that is responsive to changes in the environment ..."

2.3.1 Advantages of Mobile Agents [12][16][17]:

Mobile agents reduce network load: One of the most pressing problems facing current IDSs is the processing of the enormous amounts of data generated by the network traffic monitoring tools and host-based audit logs. IDSs typically process most of this data locally. However, abstracted forms of the data are often sent to other network locations where the data is further abstracted and then eventually sent to a central processing site that evaluates abstracted results from all location in the network. Even though the data is usually abstracted before being sent out on the network, the amount of data can still place a considerable communication load on the network. Mobile agents offer an opportunity to reduce the network load by eliminating the need for this data transfer.

i. **Overcoming Network Latency:** Mobile agents are useful for applications that need to respond in real time to changes in their environment, because they can be dispatched from a central controller to carry out operations directly at the remote point of interest. In addition to detecting and diagnosing potential network intrusions, an IDS also provides an appropriate response in order to protect and defend the network from malicious behavior. Thus cases where the central controller may have to respond to a number of events throughout the network may lead to a single point of failure. If connections to this central server are slow or unreliable, the network communications are susceptible to unacceptable delays. Mobile agents, since they are distributed throughout the network, may take advantage of alternate routes around any problem communication links.

ii. **Asynchronous Execution and Autonomy:** Mobile agent frameworks allow IDSs to continue operation in the event of the failure of a central controller or communication link. The mobile agents are launched from a home platform and can continue to operate autonomously even if the host platform from where they were launched is no longer available or connected to the network. Thus a mobile agent's inability to
communicate with central controller would not prevent it from carrying out its assigned tasks.

iii. **Structure and Composition:** An IDS can be divided into data producer and data analyzer components and represented as agents. The data producer provides an interface to the networks it sniffs or audit trails it filters. Multiple analyzers, each responsible for detecting a single attack or a small set of attacks, interact with the producer to look for attacks. The agent orientation and mobility considerations provide inherent motivation for identifying and compartmentalizing functionality.

iv. **Adapting Dynamically:** Each computing node in the network will require different tests and these tests will change over time; some tests will no longer be necessary, while new tests will need to be added to the test suite as new vulnerabilities and threats evolve. MAs provide a versatile and adaptive computing paradigm as they can be retracted, dispatched, cloned, or put to sleep as network and host conditions change.

v. **Operating in Heterogeneous Environments:** Mobile agents are generally computer and transport-layer independent, and dependent only on their execution environment. They offer an attractive approach for heterogeneous system integration.

vi. **Robust and Fault-tolerant Behavior:** The ability of mobile agents to react dynamically to unfavorable situations and events takes it easier to build robust distributed systems. Their support for disconnected operation and distributed design paradigms eliminate single point of failure problems and allow mobile agents to offer fault-tolerant characteristics.

vii. **Scalability:** The computational load on centralized IDSs increases as more processing nodes are added to the networks they monitor. Distributed Mobile Agent IDS architectures are one of several options that allow computational load and diagnostic responsibilities to be distributed throughout a network. As the number of computing elements in the network increases, agents can be cloned and dispatched to new machines in the network.

Through this chapter we have identified what a warehouse is, the issues of the warehouse and the different kinds of warehouse. As the document essentially talks about the concept of
dynamism through the warehouses and since dynamism can be implemented through mobile agents, the advantages of mobile agents have also been discusses. The next chapter discusses in detail the taxonomy of the Data Warehouse.
3. TAXONOMY AND BASIC DESIGN OF A DATA WAREHOUSE

A data warehouse is not a standalone system; it serves as a central repository/analyzer of some portion of the organizations memory. The data warehouse has to be able to perform both data integration and data manipulation functions. Thus a phrase local data warehouse refers to the portion of the data warehouse that stores the data, stores and initiates the tools and queries the data. Thus the data warehouse should be able to integrate the data sources of the organizations memory i.e. the local data warehouses and extract the portion of the data that is required to support the decision making process.

The basic taxonomy of a data warehouse can be described as a \( M \) tuple entity with each of the \( M \) parameters representing a feature of the data warehouse.

This chapter discusses each of the above parameters that make up the warehouse. Each of these parameters can be classified as essential or optional and can also be defined for the various levels that can exist. Figure 3.1 shows a data warehouse represented as a unit made up of these factors [2][3][15].

Data Warehouse= (Source/Base data, Extraction tools, Integration, Query support, User Interface, Tool Support, Backup, Updates, Mobility/location)

3.1 Taxonomy Tuple:

We briefly overview each position in the taxonomy tuple:

3.1.1 Base Source data: This defines the data source upon which the warehouse is built. This can be a simple data file without any database support, a relational database, an object oriented database, an integration of smaller databases, a data mart or even a smaller data warehouse. The base data source can be the operational database system, which is the transaction processing system for the organization.

3.1.2 Extraction Tools: The data warehouse is actually a store for information extracted from the base data thus extraction tools are required to pull out necessary information from the base data source. Thus some mechanism is needed to extract the required
information from the base data source and load it into the data warehouse. Extraction of data leads to additional issues like:

i. **Data Migration**: Case where data warehouse is in one location and the base data upon which it is operating is in some other location thus it is required that the data from the base data source be physically moved to the warehouse site. This may require sophisticated tools depending upon the size, operational complexity and compatibility of the warehouse and the data source.

![Diagram of a data warehouse as a composition of its building blocks](image)

**Figure 3.1** The model of a data warehouse as a composition of its building blocks
ii. **Data Cleaning**: Case as to how to determine which data is necessary from the data source to signify the information available from the warehouse. Thus data cleaning tools ensure that data that is not complete or does not adhere to the format or does not obey the data warehouse constraints is neglected and not extracted thus avoiding spurious data. This is an optional facet of the extraction process.

### 3.1.3 Integration tools:

The data warehouse as a whole can be thought of as the collection of different local data warehouses which collect data from different base data sources. But data in this central warehouse should be monolithic i.e. there should not be any syntactic or semantic differences. There can be different levels of integration based upon the nature of the base data source.

i. **No Integration**: The base data is monolithic and thus there is no need to perform the integration explicitly. Thus such a case arises when the base data source is data files or a single database or multiple databases with uniform syntax and semantics.

ii. **Syntactic Integration**: The base data is from multiple sources and of different format and different type. The entities and the properties are same in the entire data source but different in the format. Thus a common acceptable format needs to be derived and a set of format conversion utilities, which the integration tools will use.

iii. **Semantic Integration**: Here the different data sources have the same type and format but differs in the number and type of entities and their relationships. This is the most complex type of integration that can occur. Here integration tools have to decide what terms from multiple databases are similar by drawing correlations and then checking whether any syntactic integration is required.

Thus integration tools have to decide what type of integration is required depending upon the nature number and complexities of the base data sources.

### 3.1.4 Query Support:

The main objective of the data warehouse is to generate useful results from analyzing base data. This parameter is optional. Thus, the warehouse can be designed such that where there is no query support.
i. No Query: This is a level of query support where no queries are allowed to the warehouse. This approach is simple, as it does not have to deal with query related issues. In this approach a fixed number of tools are run on data periodically. The only output can be in the form of batch reports and alerts. This approach may be useful in data warehouses where security is of a greater concern and hence allows only a limited number of tools.

ii. Limited Query Support: The next level of query support is a limited query support where the warehouse allows only a limited set of online queries whose number is determined during system design. Thus a user cannot execute queries outside this set thus limiting his possibilities. Only the Warehouse designers can increase this set.

iii. Adhoc Queries: The highest level of query support is adhoc queries where in the user can present queries at random to the warehouse. This includes Online Analytical Processing tools that visualize the data source as multilevel database.

3.1.5 User Interface: User interface is a mechanism by which the users of the data warehouse can define the different warehouse features and access them. User Interfaces are points where the warehouse interacts with the outside world. The user interface thus makes the data and tools available to the user. Applications using massive query support require a well developed graphical user interface to carry out the processing. Applications in which the warehouse is used for some rule generation and has no query support, the interface may not be well developed and hence can go in for an application programming interface. The API defines the programs which runs continuously on the system makes use of a set of files as the base data source continuously extracts information applies the tools to get the result.

3.1.6 Tool Support: The complexity of a warehouse depends a lot on the tools in the warehouse, which again depend on the size and complexity of the data source. Tool support is an essential parameter irrespective of the type of data warehouse. These tools run on the data in the warehouse to generate useful results. A wide variety of tools can be integrated in the warehouse. Some of them are:

- Query optimization tools: Tools, which modify the user, query such that less amount of resources spent on obtaining the same results. These types of tools are
required in OLAP application where the amount of data is so large that it is required to decrease the number of disk accesses to improve performance. In case the base data source is a relational database performing mathematical operations on the underlying relational algebra can do the query optimization. Also for object oriented database the underlying algebra helps in defining the optimization strategies. New strategies have also been proposed to perform queries on XML type of data and hence to optimize the queries on such data which is simply a plain text files and unstructured and a semi structured calculus is proposed which helps in the application of mathematical formulas.

- **User Learning tools:** In the data warehouse a way to represent data is to figure out a way to represent the data which tells us if there is an intrusion or not[19]. There is some patterns which signifies that something unwanted is occurring in the system and there exists some pattern which tells that the system is functioning properly. Hence, the data is collected from different sources, represented through tools and analyzed to check out if there is an intrusion or not. Figure 3.2 gives an idea of the different ways in which the data can be represent to check for intrusions.

![Diagram of Data Representation](image)

**Figure 3.2 Different Approaches to Represent Data to Check for Intrusions.**
3.1.7 **Back-Ups:** Backups are generally used for recovery. It basically involves data replication such that in case of a failure the back up copy can define the current state of the warehouse. Several issues are again involved with this like the maintenance of consistency with in the original data warehouse and its back up. Thus depending upon the affordability, which again depends on the size and complexity of the warehouse, there can be

i. **No Backups:** If the warehouse goes down, it is simply rebuilt.

ii. **Partial backups:** Only the components of the warehouse added or modified since the last backup are to be backed up

iii. **Full backups:** Take regular back ups of the whole system inclusive of all the components that constitute the state of the warehouse i.e. data system files, tools etc.

3.1.8 **Warehouse Updates:** A warehouse contains time variant data thus a decision needs to be made as to if the data in the warehouse represents the current state of the data in the system. The various option thus need to be associated with the warehouse are

i. **No updates:** The data once loaded in the warehouse is not changed at all.

ii. **Full updates:** Allows the entire data in the warehouse to be changed in case of a change in the base data.

iii. **Partial updates:** Allows reloading of a selected portion corresponding to whose underlying base data has been changed.

3.1.9 **Mediator Systems:** In some cases the base data source is not conventional data. It may contain images and other multimedia data. In such cases a mediator system that helps bridging semantics is useful. This support is required when external or multimedia data is used as base data. This is not required for every data warehouse application and hence is not an essential parameter. The mediator helps in keeping the data in the data warehouse monolithic.

3.1.10 **Location:** Indicates the physical location of the warehouse. Most warehouses are used for information retrieval and thus serve only as a repository. Hence, most of the warehouses have a certain fixed physical location. However, this is not a necessity. A warehouse that is not confined to a fixed physical location can be built. Thus evolves the concept of dynamic data warehouses. This parameter has been added to the
present the warehouse that differs significantly from the traditional data warehouses in terms of purpose and application.
4. DYNAMIC DATA WAREHOUSE MODEL

In chapter 3 all the parameters, which essentially embody a data warehouse, are listed. Using the taxonomy listed in the previous chapter, this chapter realizes the essential and optional features of the warehouse and comes up with the concept of a light weight dynamic data warehouse.

4.1 Dynamic Data Warehouse Design:

A "light weight Dynamic Data warehouse"[3] satisfies all the essential requirements for a warehouse. The use of the word dynamic signifies that it does not have a fixed location and hence it is built using mobile agents thus essentially using their advantages in building a more effective system. Such a warehouse can be constructed and destroyed when the user wants.

The basic underlying taxonomy of this warehouse is similar to the taxonomy mentioned in the previous chapter. The design of this warehouse is shown in Figure 4.1. This warehouse thus considers only a few features from the previous taxonomy rest uses as optional thus has an N tuple structure. The justification regarding the inclusion of these features is also presented. The different parameters used in this design are:

Dynamic Data Warehouse = (Source/Base data, Query Support, User Interface, Tool Support, Mobility/Location)

4.1.1 Interface: Every data Warehouse must have some way of interacting with its users. This is an essential parameter as discussed in previous chapter. Options available for the user interaction with the warehouse are:

i. GUI: Graphical User Interface: This is a full fledged user friendly interface that allows the users of the data warehouse a range of tasks such as Querying the warehouse, running the available tools on the data and adding the tools to the warehouse.
ii. **API**: Application program Interface: Less user friendly approach in which the operations on the warehouse are predefined and the user is restricted to the usage of only those predefined set of operations.

iii. **No Interface**: Case where the warehouse does not operate on any interactive mode but operates on a batch mode thus a set of tasks are carried out on a periodic basis without any user intervention.

![Figure 4.1 Proposed design of dynamic data warehouse.](image)

**4.1.2 Data Storage**: This feature is also essential to any data warehouse as discussed in chapter 3. The base data can be in the form of files or a normal database.
i. File: The file is far more simplistic notion of data store. Files are a source of uniform and integrated data. The semantics of the files can be defined in a file handler or wrapper. This data store is useful when the user wants to gather information from audit trails or system files specifically for intrusion detection.

ii. Database: This is more developed type of data source. A database mostly concentrates on operational data. The database can in turn be a relational or object oriented one.

4.1.3 Query Support: The design offers a limited query approach where the user can execute only a pre-specified set of queries. Queries in this approach are included in the form of tools where in the tools can be used to execute queries on a periodic basis or on user demand. This in turn leads to the requirement of a user interface if the tools are to be executed on user demand.

4.1.4 Data Tool Support: A list of tools, which operate on data after it, has been loaded in the warehouse. These tools often relate closely to the application for which the warehouse is being built.

   i. Commercial Data: commercial data processing requires how efficiently data is stored as here we are dealing with a large amount of data. it also deals with how effectively can the data be extracted from the warehouse. Thus several data mining algorithms can be implemented as tools. These tools operate on data and generate useful results known as tools.

   ii. Geo-Spatial Data: The kind of treatment for geo-spatial data is entirely different as over here we have an image driven database. Thus tool for conflation and superposition can be used in the data warehouse.
5. IMPLEMENTATION ISSUES

Chapter 3 presents a prototype of the various components involved in the taxonomy of a data warehouse. Chapter 4 discusses in detail what exactly is meant by the term dynamic with respect to a data warehouse perspective and presents a conceptual model of such a system. The model is presented as taxonomy. In this Chapter a detailed description of the implementation of the dynamic data warehouse is presented through the use of screenshots and class descriptions.

5.1 Dynamic Data warehouse Prototype:

The implementation of this dynamic data warehouse involves four parts. Architecture of the prototype is presented in Figure 5.1.

5.1.1 Console: This is the main engine for the dynamic data warehouse represents the graphical user Interface. Helps in selection of the different parameters corresponding to the data warehouse such as selection of base data source and availability of query support etc. The console also allows the selection of tools for the warehouse. The console then fetches the appropriate set of agents and passes control to the interface agent.
5.1.2 Interface Agent: This agent creates the user Interface as specified by the console on creation of the warehouse. First function performed is to get the user interface details from a distant location and then use this interface to communicate with the user. This agent interacts with the tool agent and the store agent to process user needs. It contains java AWT component information for the GUI interface. On selection of GUI type of interface this agent gets these AWT components and the methods to display the same. On selection of API type of interface, it provides methods to access the warehouse from an external program. On having arrived at its destination, this agent informs the destination about its arrival and its type. This indicates that the warehouse is ready for operation. This agent has the information of the API provided by the data warehouse. Interface agent interacts with the storage agent for normal queries and with the tools agent for tool operation.

5.1.3 Store Agent: This agent is also selected by console. This agent is responsible for obtaining store details. This agent should provide a clean interface to the Interface agent and Tools agent isolating them from the underlying details of the data store. This agent contains the JDBC routines required for a database kind of a store. Otherwise it contains the routines to read through the data file and extract information. For this it needs to have information about the file structure. It provides basic record fetching routines for the interface agent and the tool agents. Thus all kinds of data stores can be accessed through this agent.

5.1.4 Tool Agent: Again the console selects these agents. Each agent represents one tool. These agents are predefined with all their methods and variables. Hence their interaction with other agents like the interface and the storage agent is fixed. It should provide a kind of a bridge between complex demands from the interface agent to the primitives provided by the storage agent. Adding a new tool for a particular data store requires writing one method in the Tool Agent.

5.2 Detailed discussion of the system:

5.2.1 Implementation Screenshots from Prototype
Step1: The basic user interface, which the console provides, is shown in Figure 5.2. It is basically a graphical user interface, which allows the user to select parameters. It is the job of the console to fetch the appropriate set of agents and pass control to the interface agent. Depending on whether at the remote location the interface for the communication should be graphical user or not there is an option corresponding to graphical user interface and application programming interface. Also depending on the base data source there is the option of selecting either a database or a file

![Dynamic Data Warehouse Console](image)

Figure 5.2 The Console used for creating and destroying the warehouse

Step2: Through this GUI the user can make the selections. Here is the scenario when the user selects the type of user Interface to be GUI, the data source to be a simple database with
added query support and tools. Now the console gets the appropriate set of agents and sends them to the remote location passing control to the interface agent as shown in Figure 5.3. The status bar gives the updates on the status of the system whether the agents have been launched successfully or not.

![Dynamic Data Warehouse Console](image)

Figure 5.3 Selection of parameters and the agents sent to the remote location

Step3: At the remote location the control is with interface agent, which creates the user interface as selected by the user on the console. Now since the GUI option was selected it generates a graphical user interface. It then interacts with the store agent and the tool agent to process the user needs. The GUI is shown in Figure 5.4. The database being used is a simple Access database. The different fields in the database are shown in the drop down menu and the different tools selected at the console are also shown in a drop down menu. Also there is
a section, which allows querying the database. The different fields in the database are shown as a drop down menu and the value to be queried is entered in the value text box. Select the field; select the tool to be executed and press RUN TOOL to get the result of the execution of the tool.

Figure 5.4 The GUI built by the Interface Agent In response to the information from the console
Figure 5.5 shows the execution of one of the tools over a field of the database.

Using the GUI built by the interface agent at the remote location the database can be accessed and queries can also be executed against the database. Figure 5.5 shows the execution of the query "the name of the cars whose price is greater than 40000" against the database.
Figure 5.6 Execution of the query against the database

Step 4: The next scenario is -- the user selects the type of user Interface to be GUI, the data source to be a simple data file. This selection is shown in figure 5.7. Since the basic underlying assumption for the files is that they are the log files, which should be checked, for intrusions hence no query support is provided for them. Over here a simple operation is being done where in the contents of the file mentioned in the pop up box (as shown in Figure 5.8) are copied into another file. The information about this existing file is provided through a header file, which is the filename, suffixed with "header". The contents of this header file are printed out after successful execution as shown in Figure 5.9. Applying different tools specific to intrusion detection and storing those rules in a new file can extend this concept of files.
Specific tools related to intrusion detection can be added later. The effort needed is to write an agent corresponding to each of these tools. The agent is the program, which carries the functionality of these tools over to the remote location where the warehouse is built and used for monitoring host data. The option for the selection of the specific tools corresponding to the need limits the number of agents to be sent across the network thereby minimizing the traffic across the network. This is in accordance with one of the requirements of an ideal intrusion detection system as mentioned in chapter 2. At any point to destroy the warehouse environment the close button can be used. Also to make a new selection of the parameters reset button can be used.
The file name is entered and the operation COPY is done such that the contents of this file is copied to a new file the new file has the same name with a suffix new. To browse through the file we need to know information about the file and this information is stored in a header file which is also name as the oldfile but with a header suffix.

Step 5: The next scenario is when we select API instead of GUI as the user interface. Since this is a less user-friendly approach, it involves the warehouse to redefine the application of tools or the queries and the output of the operations is printed out at the command prompt. The API part of the warehouse is particularly useful in the intrusion detection part of the warehouse where in we want to run certain operation constantly at the background without user interference. The output is shown in figure 5.10.

The API kind of an interface is particularly useful when the system is required to do some operations in the background periodically with out any user intervention.
In this case the first four entries specify the different parameters received by the interface agent specifying the user interface to be an API and the base data source to be a database and the tools involved. The next three specify the result of the queries and tools against the database. The first is the result of the query asking the names of the car models released in 2001. The second and the third results are the application of the tool ADD on the fields’ price and rating.

Figure 5.10 Results of the query and tools when the type of interface selected is an API
6. EVALUATION

Chapter 4 presents the model of a Dynamic Data Warehouse and chapter 5 goes in depth about the implementation details of this warehouse in terms of class definitions and screen shots. Now that an introduction to the warehouse is actually done an issue which needs to be addressed is the areas where the use of a dynamic warehouse is most effective. Here we need to understand the tradeoff between efficiency and correctness. There might be some applications which might use both a static and a dynamic warehouse with no difference as such the use of dynamism may not be actually useful for such applications. Whilst there are some applications where the use of a dynamic warehouse can for sure improve the performance and correctness of the system. Section 6.1 talks about some of the areas where a dynamic data warehouse can be of significant importance. Also in the warehouse model defined there are some issues, which need to be addresses for ensuring the robustness of the model. Section 6.2 talks about some of those issues and some ideas as to how these issues can be efficiently removed from the design. It again focuses attention towards some other issues of data processing and distributed network management.

6.1 Application Area Examples:

While discussing warehouses and the concept of “Dynamism” in the warehouse the question that should be directly focused is the problem areas where the mobile nature of the dynamic data warehouses approach can be significantly used. Two areas where the concept can be used and application using this concept is discussed in the section.

6.1.1 Moving Computation and Tools:

In many applications the location and the amount of data needed at that location can vary rather dramatically. In cases where the amount of data is large, it is more reasonable to move the tools to the data source rather than to move the data to the tools. In particular there can be several applications of this type in areas such as bringing geo spatial data to field users [21]. Allowing the warehouse to move to the data site or at least a site closer to the data can
greatly enhance the effectiveness of such a system. In particular it reduces the network flow and allows the system to balance the processing load across the network.

6.1.2 Reducing Warehouse visibility:

In security applications a data warehouse can be very beneficial, but a static warehouse provides another potential target for a hacker. A data warehouse is very useful in intrusion detection systems defined in Chapter 3. A warehouse can be used to develop rules for detecting changes in the system behavior. The problem is that a static warehouse provides a target for hackers wanting to undermine the effectiveness of the Intrusion Detection System.

A hacker that can gain access to the data warehouse can poison the data stored in there. The result is that the generated rules can be modified so that the Intrusion Detection System sees every normal action as intrusion. This effectively shuts down the network. Perhaps even more dangerous, the hacker can seed the data with operations that allow the hacker freedom in developing undetected attacks.

The Dynamic data warehouse lessens the likelihood of a hacker getting access to the data by reducing the visibility of the data warehouse. Its location can be changed and decoy data warehouse can be set up to make it difficult for a hacker to get access to the warehouse.

6.2 Issues in the Dynamic Data Warehouse:

6.2.1 Storage capabilities of warehouse

One of the issues with the Dynamic warehouse is its storage capacity. The storage capacity of the Dynamic Data Warehouse is specified as an optional parameter. Thus the warehouse can only have a set of tools running on remote database with out any of its local storage. Such a warehouse can only be used for misuse detection. Also there can be the warehouse, which has data cleaning tools, which extract only the required information, store it in the local storage of the database and then apply tools and again store it in the local store. Any new information obtained in this process is stored in the local store. A decision needs to be done as to how and when this local information is updated. How essentially gives a choice of copying the whole information in the local data store when the warehouse has done all its
work and is being destroyed. Also there can be a choice of updating the central store with the information in the local store after some specified interval of time. This time slice however needs to be decided at the design phase and differs depending on the application. Also the issue is where this information is to be updated. If it is stored in some global storage space, which might act as a repository to rules, then efforts need to be made to save this store from the intruder. Such a central store can again be vulnerable to data poisoning where while entering the rules itself little manipulations are done which though are below that threshold may still have the cumulative effect of ignoring an intrusion.

6.2.2 Information management

Another issue, which needs to be addressed particularly for intrusion detection, is information management. The Warehouse sends agents to remote locations, which do operations and send in the rules back to the central repository for rules. The issue over here is which rules in this repository are to be denoted as outdated and expunge and which of them as current. There needs to be some algorithm to signify the information in the warehouses' central storage. Also an algorithm is needed which determines which information is to be removed in order to add new information to the central repository. When the phrase “destroying the warehouse at the remote location” is used it essentially mean that all the information, which has been gathered by the warehouse through the execution of the tools, has to be stored somewhere for analysis. Again an attack can be distributed over days and hence the rules generated over these days might all be needed to determine the cumulative effect, which might be an intrusion. So though we say that the Dynamic Data Warehouse is more efficient as it moves computation to the remote location the management of data in the central repository still continues to be a problem as in a static data warehouse.

6.2.3 Distributed control in Intrusion Detection

It is assumed that through the dynamic data warehouse the server sends the agents to different locations, which might gather information and come back to a central location for analysis. There are attacks on the system, which might be distributed in nature. Thus to actually detect the attack it is necessary to correlate all the information. At this point again
the issues in distributed programming come into picture. What is required is the action of the system at a particular point of time. But in distributed system the units are at separate locations with a different time clock. Thus what might be the current information might be the over dated information for another system. It might be possible that these two events in different systems overall contribute to an intrusion. It is therefore necessary to figure out an algorithm, which helps in overcoming all the discrepancies of the clock. Thus the concept of a global clock needs to arise.
7. INTRUSION DETECTION SYSTEM MODEL

In this chapter first the requirements from a generic intrusion detection model are discussed. Then an intrusion detection system architecture will be briefly discussed. It makes use of mobile agents and has a database to store information, which can in turn be used to detect intrusions. Also there is an overview of how mobile agents satisfy some of the requirements of the ideal intrusion detection model. In the later part of the chapter there is a brief analysis of the system in which a data warehouse replaces this database and how a data warehouse solves some of the issues associated with a mobile agents intrusion detection system with a database.

Nearly all present day commercial IDSs follow a hierarchical architecture. Information gathering occurs at leaf nodes, network based or host based collection points. Event information is passed to internal nodes that aggregate information from multiple leaf nodes. Further aggregation and abstraction and data reduction can occur at higher internal nodes until the root node is reached. The root is a command and control system that evaluates attack situations and issues responses. The root typically reports to an operator console where an administrator can manually assess status and issue commands.

There are lots of issues in the current intrusion Detection systems which can be solved through the use of a mobile system architecture.

7.1 Mobile Agent Intrusion Detection System: MAIDS[4][12]

MAIDS is an implementation of the concept of Distributed Intrusion detection systems where in a there is a distributed monitoring system as opposed to a central monitoring system giving it the additional capabilities of detecting distributed attacks. The MAIDS architecture is characterized by certain well defines layers. Agents have been developed for the system that retrieve information from distributed systems, classifies the data using embedded expert rules or machine learning techniques, and stores data in a database. The fundamental design of the Intrusion detection system is illustrated in Figure 3.1.
Figure 7.1 Architecture of the Mobile Agents Intrusion Detection System
At the bottom of the tiered architecture, the system log routers and system activity agents read log files and monitor the operation of the systems. The routers feed into the distributed data-cleaning agents, which previously registered their interest in particular, events. Targeted data cleaning agents process data obtained from the routers and activity agents. They render data into common data formats.

In the middle of the architecture the low level, mobile agents form the first line of intrusion detection. They periodically travel to each of their associated data cleaning agents, obtain the recently gleaned information, and classify the data to determine whether singular intrusions have occurred.

The low level agents are managed by mediators which control the systems visited by agents, obtain classified data from the agents, and route the data into the local database and to the user interface. As the system is developed, the mediators will apply data mining algorithms to the data in the database to connect individual events into a cohesive view of the elements involved in an attack. At the very top of the hierarchy comes the interface agent, which serves as the analyst console.

The unique feature of MAIDS is that it is developed using a disciplined requirements engineering process. The entire process comprises of three stages:

- A high level description of the system vulnerabilities is created using software fault trees (SFT)
- This description is then transformed into a Colored Petri Net (CPN) which serves as the IDS design specification.
- The CPN is then transformed into the actual implementation of MAIDS agents.

### 7.2 Mobile Agent Intrusion Detection system with a data warehouse:

Some of the issues discussed in this section are – where exactly does a data warehouse fit in an intrusion detection system environment and what is the use of such a warehouse for this intrusion detection system, essentially the merits of a data warehouse over a simple database.
Also what are some of the disadvantages of using a warehouse and how modifications over the basic structure of a data warehouse helps in overcoming these shortcomings and development of a more robust system. [18]

7.2.1 Intrusion Detection System with a Static Data Warehouse:

In the mobile agent intrusion detection system as discussed in the previous section a high level agent called mediator is used. A mediator is a high level agent, which interacts with the database. It obtains data from the different low-level agents and stores the data in the database. The data in the database should be monolithic and hence the mediator has to convert the data from different sources into a common acceptable format. If the term “wrapper” is used to denote a program, which converts data from one format to another, the mediator interacts with many such wrappers to convert the available data in different formats into a common acceptable format. Figure 3.2 shows such a scenario.

Figure 7.2 A part of the IDS shown in detail with the use of wrappers.
To essentially facilitate the interaction of the wrapper with the mediator another file or program might be needed to relate one particular data format to another. Also as the system is developed the mediators will apply data mining algorithms to the data in the database. Again the output of these algorithms needs to be converted to a format consistent with the database. So additional wrappers and files are needed to serve the purpose. One requirement of an ideal intrusion detection system is “Less burdensome maintenance” i.e. the maintenance of the IDS should take minimal effort in terms of resources and manpower. Keeping track of so many tools corresponding to different algorithms and format conversions is a cumbersome process. Thus if we have a single warehouse which encompasses the database (which is the repository of useful information) and all the other tools which might act on this database then the process of maintenance of different tools and wrappers becomes minimal.

The problem here is abstracted by visualizing the data warehouse as one unit ignoring the details of the warehouse in terms of application of tools, data format requirements for different tools, and data format conversions. The data warehouse can be represented in terms of database and tools ignoring the underlying details.

7.2.2 Need of a Dynamic Data Warehouse:

Another feature of an ideal Intrusion Detection System is that it should not be vulnerable to direct attack. Instead of attacking the system the intruder can attack and compromise the IDS itself thereby manipulating the rule generation process and the rule comparison process. A pattern, which relates to an intrusion, can now be represented as a harmless system activity. As mentioned a data warehouse has the base data for analysis and tools for rule generation and matching. On the whole, the data warehouse can become a central point of attack if the intruder can get hold of the location of this static warehouse. In such scenarios the concept of dynamism is more useful. Figure 3.3 shows the working of such a dynamic data warehouse.

The warehouse can be constructed at the remote location by defining the different components of the warehouse, sending the tools in the form of agents to the remote location
and building the warehouse over there. Thus, there is not a single central location of the warehouse, which can become a direct target. The location of the data warehouse varies

**User perception**
- Select tools and the base data source
- Does not have to think about the underlying complexities of wrappers

**Remote host**
- Agents corresponding to the different parameters of the warehouse, building the warehouse at the remote location
- Base Data source which can be file or database

**Central server with datawarehouse**
- System files for use of mediator
- Mediator
- Wrappers
- Database
- Tools

Figure 7.3 Working of a dynamic data warehouse.

Depending on the location of the remote host where monitoring is done. Also the warehouse can be defined with no storage or limited storage capacity.
A data warehouse with no storage capacity takes the knowledge defines the base data source for monitoring and applies the tools and uses the knowledge for analysis. There is no updating of information and hence the warehouse can be created and destroyed at the remote location. Also there can be limited storage capacity as the warehouse can gather and analyze the information through the application of tools and bring the information back to the central server.
8. CONCLUSION

The paper explores the possibility of a warehouse that is not confined to a location. The prototype presented in chapter 5 is only a simplistic version of the dynamic data warehouse. The model has been tested by building a prototype. The most important achievement is the demonstration of the fact that data warehousing can be lightweight activity and can be dynamic. There can be various applications where such a warehouse can be useful.

Intrusion detection is one such area and the current work in using the warehouse efficiently can be to integrate it with one of the applications where the use of dynamism has enough significance.

Future work includes increasing the range of tools that can be supported by the warehouse thus catering to an increased number of applications. Also it would significantly make the warehouse more robust and efficient if the issues on data management and distributed control are addressed and handled effectively.
REFERENCES

(Date accessed: May 23, 2002)

[2] Yeping Zhou. A Data Warehouse Based on Materializing Object-Oriented Views; Creative Component, Department of Computer Science, Iowa State University, December 2000.
(Date accessed: May 23, 2002)

(Date accessed: Aug 13, 2001)

(Date accessed: May 23, 2002)


(Date accessed: May 23, 2002)

(Date accessed: May 23, 2002)

(Date accessed: May 23, 2002)

(Date accessed: May 23, 2002)


(Date accessed: May 23, 2002)

(Date accessed: May 23, 2002)

(Date accessed: May 23, 2002)

[18] Kai Zhu: Data Warehouses Support for Intrusion Detection; Creative Component, Department of Computer Science, Iowa State University. (Date accessed: May 23, 2002)


Voyager

ObjectSpace Voyager is the ObjectSpace product line designed to help developers produce high-impact distributed systems quickly. Voyager is 100% Java and is designed to use the Java language object model. Voyager allows usage of regular message syntax to construct remote objects, send them messages, and move them between programs. Voyager is fully compatible with JDK 1.1 and JDK 1.2. Voyager offers developers a complete set of features for distributed application development, including:

i. **Remote-Enabling of Classes:** Java classes are automatically remote-enabled at runtime. A class does not have to be modified in any way, and no additional files are necessary to remote-enable a class. Thus, there is no difference between a "regular" Java class and a remote-enabled class. Classes may also be explicitly remote-enabled by declaring them to implement either `java.rmi.Remote` or `com.objectspace.voyager.IRemote`.

ii. **Remote Object Construction:** With a single method call, you can create a remote instance of any class on any Voyager VM and obtain a proxy to the newly created object. The proxy class is generated dynamically if it does not exist, eliminating the need to manually generate this class. Since the proxy implements the same interfaces as the object, by using interface-based programming techniques you do not need to modify any code to work with remote objects.

iii. **Dynamic Class Loading:** Classes can be dynamically loaded from one or more locations when necessary. This allows you to easily set up class repositories that serve your corporate Java applications, simplifying deployment and maintenance.

iv. **Remote Messaging:** Method calls to a proxy are transparently forwarded to its object. If the object is in a remote program, the arguments are serialized using the standard Java serialization mechanism and de-serialized at the destination. The morphology of the arguments is maintained. If an object's class implements `com.objectspace.voyager.IRemote` or `java.rmi.Remote`, the object is passed by reference. An appropriate proxy class will be generated dynamically if needed. If an
object's class implements `java.io.Serializable`, it will be passed by value. Objects, which implement none of these interfaces, are passed by reference.

v. **Object Mobility:** You can easily move any serializable object between Voyager programs at runtime. Voyager automatically tracks the current location of the object. If a message is sent from a proxy to an object's old location, the proxy is automatically updated with the new location and the message is re-sent. Mobility is often useful when optimizing message traffic in a distributed system.

vi. **Autonomous Mobile Agents:** Voyager supports mobile, autonomous agents that move themselves between programs and continue to execute upon arrival at a new location. It is easy to build agents that move between Voyager VM's to more efficiently satisfy their goals.

vii. **Naming Service:** Voyager's naming service provides a single, simple interface that unifies access to standard naming services such as RMI, CORBA, and JNDI. New naming services can be dynamically plugged into Voyager's naming service.

viii. **Distributed Garbage Collection:** The distributed garbage collector automatically reclaims objects when there are no more local or remote references to them. This eliminates the need to explicitly track remote references to an object. The DGC mechanism uses an efficient "delta pinging" algorithm to minimize the traffic required for distributed garbage collection. You can also fine-tune the behavior of the distributed garbage collection mechanism and receive notification of DGC events.

ix. **Dynamic Aggregation™:** This feature allows you to add secondary objects (termed facets) to a primary object at runtime. For example, you can dynamically add hobbies to an employee, a repair history to a car, or a payment record to a customer. Dynamic aggregation represents a fundamental step forward for object modeling and complements the traditional mechanisms of inheritance and polymorphism.

### A.1 Starting and Stopping a Voyager Program

A program must invoke one of the following variations of `Voyager.startup()` before it can use any Voyager features:
• **startup()**

Starts Voyager as a client that initially does not accept incoming connections from remote programs.

• **startup( String url )**

Starts Voyager as a server that accepts incoming connections, either on the specified URL or on a random unused port if the URL is `null`.

**A.2 Dynamic Aggregation:**

Voyager supports dynamic aggregation, which allows you to attach new code and data to an object at runtime. Dynamic aggregation allows you to attach secondary objects, or facets, to a primary object at runtime. A primary object and its facets form an aggregate that is typically persisted, moved, and garbage-collected as a single unit. A primary object's facets are represented by an instance of `Facets` that is initially set to `null`. To access an object's `Facets`, we can use one of the following static `Facets` methods:

• **get( Object object )**

Returns the object's `Facets`, which may be `null`.

• **of( Object object )**

Returns the object's `Facets`, setting it to an initialized instance of `Facets` when it is currently equal to `null`.

**A.3 Naming Services:**

The Voyager Namespace service provides unified access to a variety of naming services. The class `com.objectspace.voyager.Namespace` is a facade, which unifies binding and lookup operations to any naming service implementation. To bind a name to an object, invoke `Namespace.bind()` with the name expressed as an URL. All proxy classes extend `com.objectspace.voyager.Proxy`. If a proxy class does not already exist, the Voyager classloader generates it dynamically. We can use any of the following methods to obtain a proxy to an object:

• **Factory.create( String classname, String url )**
Returns a proxy to a newly created object, where *classname* is the name of the class that you are creating an instance of, and *url* and *url* specifies where the object should be created.

- *Namespace.lookup(String name)*

  Returns a proxy to the object with a particular name.

- *Proxy.of(Object object)*

  If the specified object is already a proxy, returns the object; otherwise returns a proxy to the object.

  A method call on a proxy is forwarded to its associated object unless it is one of the following special methods:

  - *getClass(), notify(), notifyAll(), wait()*

    These methods are all final methods in *Object* and are executed directly by the proxy.

  - *equals()*

    Returns true if the argument is a proxy that refers to the same object as the receiver.

  - *isLocal()*

    Returns true if the proxy is in the same VM as its associated object.

  - *getLocal()*

    If the proxy is in the same VM as its associated object, returns a direct reference to the object; otherwise returns *null*

  - *getURL()*

    Returns the URL of the proxy's associated object

---

**A.4 Mobility and Agents:**

Mobility allows you to move objects that exchange large numbers of messages closer to each other to reduce network traffic and increase throughput. To move an object to new location, use *Mobility.of()* to obtain the object's mobility facet and then use the methods defined in *IMobility*:

- *moveTo(String url)*

  Moves to the program with the specified URL.

- *moveTo(Object object)*
Moves to the program that contains the specified object.

To make an object a mobile autonomous agent, use Agent.of() to obtain the object's agent facet and then use the methods defined in IAgent:

- **`moveTo(String url, String callback[, Object[] args])`**
  Moves to the program with the specified URL and then restarts by executing a oneway callback with optional arguments.

- **`setAutonomous(boolean flag)`**
  If the flag is true, become autonomous. An autonomous agent is not reclaimed by the garbage collector even if there are no more local or remote references to it. An agent is initially autonomous by default, and typically executes `setAutonomous(false)` when it has achieved its goal and wishes to be garbage collected.

- **`isAutonomous()`**
  Return true if this agent is autonomous.

---

**A.5 Distributed Garbage Collection:**

Voyager Supports distributed garbage collection. The distributed garbage collector reclaims objects when there are no more local or remote references to them.

```
Agent.of(this).setAutonomous(False);
```

The agent kills itself by executing this statement.