# WebRACE: A Distributed WWW Retrieval, Annotation & Caching Engine

**Marios D. Dikaiakos**
Department of Computer Science
University of Cyprus
mdd@ucy.ac.cy

**Demetris Zeinalipour-Yiazti**
WinMob Technologies
Nicosia, Cyprus
csyiazti@ucy.ac.cy

*PADDA 2001*
*Munich, April 19-20, 2001*

## Outline of the Presentation

- eRACE Project: Context and Goals
- eRACE Infrastructure: An Overview
- WebRACE: Architecture & Implementation
- Current Status and Future Work



## The Context

The paradigm shift of the basic Web-services model:

- From that of a Web-server running on a well-defined host and providing content to clients over a specific communication protocol (HTTP)
- To a fully distributed and dynamic web of interacting servers and software entities, possibly mobile, deployed at a global scale, serving a variety of terminals with widely differing capabilities





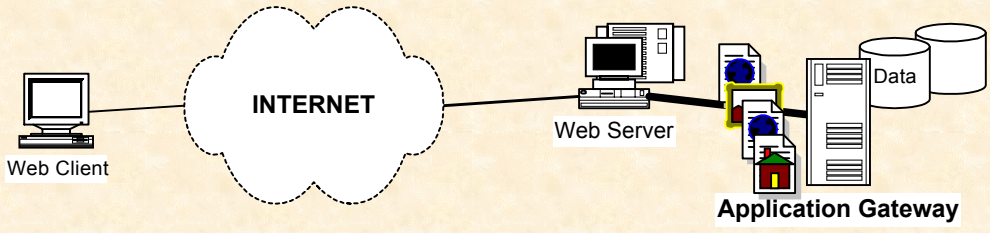
## From Web Servers to Web Services: I




- *Typical client-server model*
- *Web server: repository of multimedia content*



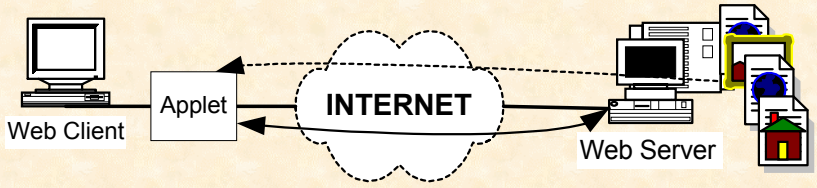
# From Web Servers to Web Services: II




- *Typical client-server model*
- *Web server: provider of dynamic content*

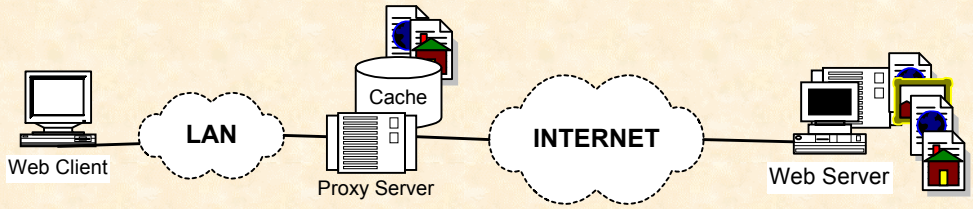


# From Web Servers to Web Services: III




- *Client-server model with dynamically enhanced clients*
- *Web server: repository of content & functionality*

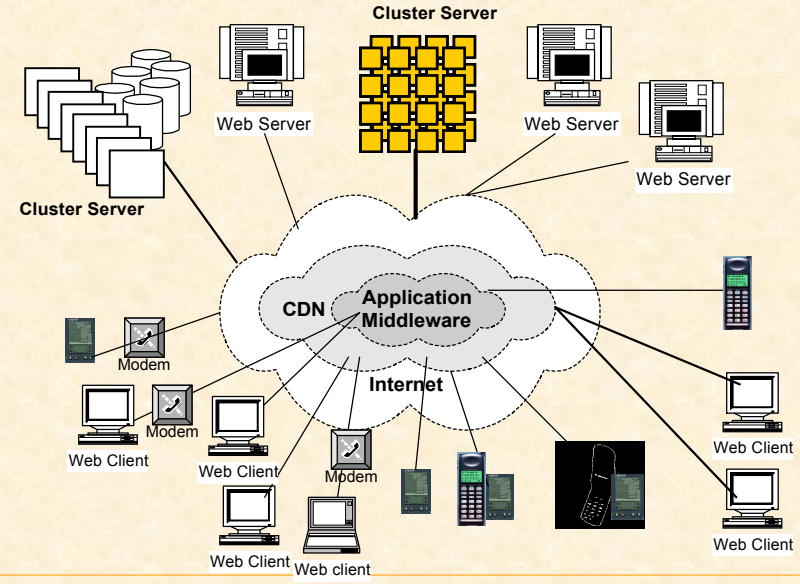
# From Web Servers to Web Services: IV



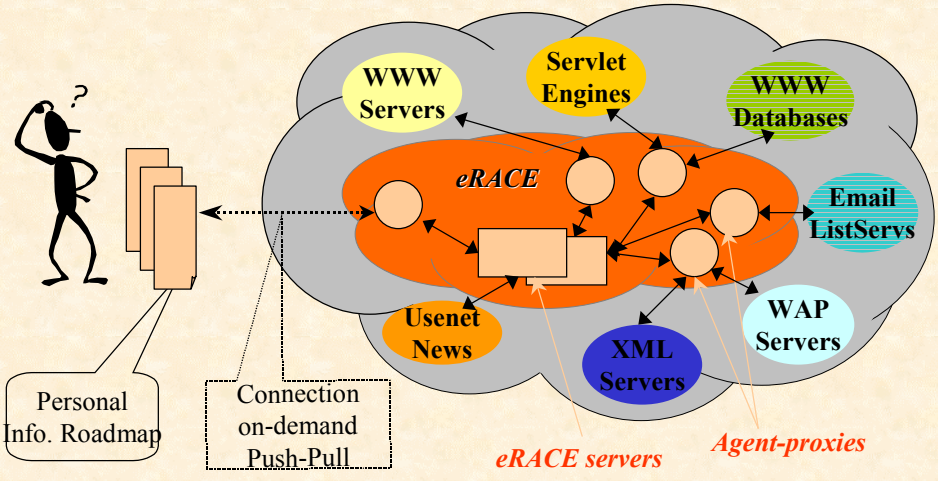

- *Proxy-server model*
- *Web caching, prefetching, information dissemination*
- *Content-Distribution Networks*



# Next-generation Internet Services



## eRACE Infrastructure Overview






## eRACE Project Goals

To develop an infrastructure that:

- Collects, transforms, customises and personalizes information from heterogeneous sources on a continuous basis, according to user interests.
- Selectively feeds information to users adapting to:
  - User interests and priorities.
  - The urgency & relevance of collected information.
  - Available connection modalities, terminal devices and preferred information-access modes.
- Provides a user-centric view of the global information space by aggregating customised content and using a simple information provision paradigm.

## eRACE Project Goals

To develop an infrastructure that:

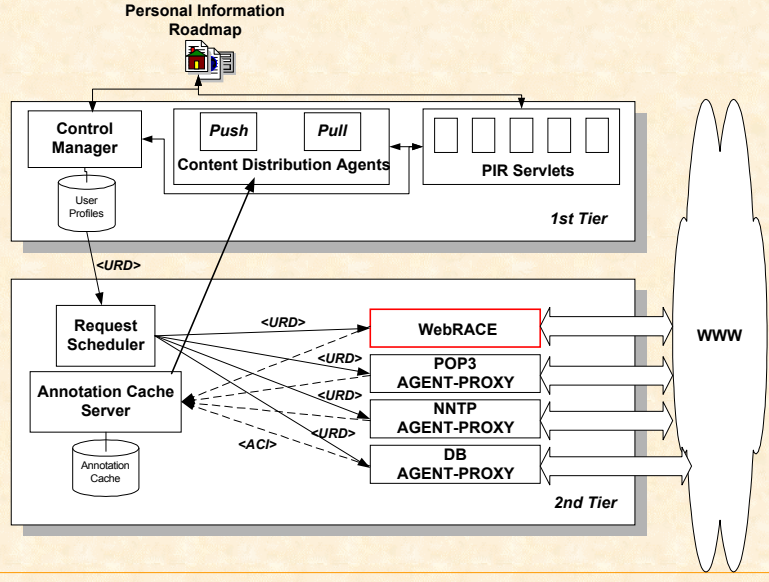
- Is incrementally scalable and can be distributed to different machines.
- Exposes policies of scheduling user-requests, QoS, garbage-collection.
- Enables the easy development of new services and re-targeting to new terminals.


## eRACE System Architecture

Two-tiered Architecture:

- Tier 1:
  - Control Manager
  - Content Distribution Agents
  - Personal Information Roadmap Servlets
- Tier 2:
  - Request Scheduler
  - Distributed Agent-Proxies (WWW, NNTP, POP3, etc.)
  - Annotation Cache Server


# eRACE System Architecture






# eRACE Information Architecture

- Information Architecture: describes the data representations of *state information* and *information exchanges*, in terms of XML DTD's:
  - Control Manager DTD: account, authentication, connection-status.
  - User Profile DTD: personal data, notification addresses, resource information (URD).
  - Annotation Cache Interface DTD: meta-information for collected content.
- Data sharing between various components is done using pass-by-value semantics (messages and events).
- This choice enables us to decouple and physically separate components.

## eRACE Information Architecture (ctd')

User-profile DTD encodes:

- Personal data
- Notification addresses (email, mobile phone)
- Resource information:
  - Resource description
  - Query options
  - User interests (keywords)
  - Notification Priorities

```
<!ELEMENT personal (name,surname,email,phone?,fax?,mobile?)>
```



## eRACE Information Architecture

Users Manager DTD encodes:

- Account & authentication information for eRACE users
- Connection status

```
<!ELEMENT Accounts (Account*)>
  <!ATTLIST Accounts id ID #REQUIRED
                     location CDATA #REQUIRED
                     maxAccounts CDATA #REQUIRED
                   locked (false | true) "false">

              <!ELEMENT      Account   EMPTY>
              <!ATTLIST      Account   id ID #REQUIRED
                                       state  (false | true) "true"
                                          docbase CDATA #REQUIRED>
```



# eRACE Information Architecture (ctd')

- Unified Resource Description (URD):

```
<!ELEMENT source (uri,      type,      keywords?,   depth?,   urgency)>
<!-- Source Information  -->
<!ELEMENT uri (#PCDATA)>
        <!ATTLIST uri group CDATA #IMPLIED
                        login CDATA #IMPLIED
                          password CDATA #IMPLIED
                        port CDATA #REQUIRED
                          timing CDATA #REQUIRED
                          lastcheck CDATA #REQUIRED>
<!ELEMENT type EMPTY>
        <!ATTLIST type protocol (http | pop3 | nntp | rmi) "http"
                                  method (push | pull) "pull"
                      processtype (filter | nonfilter) "filter">
<!-- Processing - Filtering Info -->
<!ELEMENT keywords (keyword+)>
          <!ELEMENT keyword EMPTY>
                    <!ATTLIST keyword key CDATA #REQUIRED   weight (1 | 2 | 3 | 4 | 5) "3">
        <!ELEMENT depth EMPTY>
                <!ATTLIST depth level (1 | 2 | 3) "1">

<!-- Urgency -->
<!ELEMENT urgency EMPTY>
                    <!ATTLIST urgency urgent (1 | 2 | 3) "2">
```



# eRACE Information Architecture (ctd')

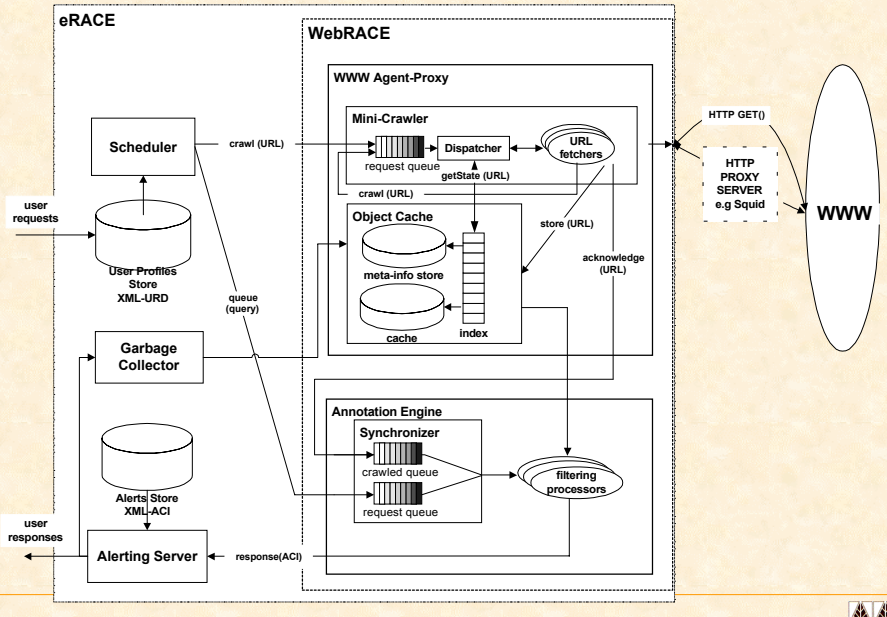
- Annotation Cache Interface (ACI): maintains structural & semantic information about collected content

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT cache (annotation+)>
<!ATTLIST cache location CDATA #REQUIRED
                size CDATA #REQUIRED
                maxsize CDATA #FIXED "50000"
                locked (false | true) #IMPLIED
                  unique_id CDATA #REQUIRED>
<!ELEMENT annotation (uri,urgency,docbase,expired,summary)>
  <!ATTLIST annotation id ID #REQUIRED
                        owner CDATA #REQUIRED
                        extension CDATA #REQUIRED
                        format (text | html | binary | multipart | unknown )
                        relevance CDATA #REQUIRED
                        updatetime CDATA #REQUIRED
                        filesize CDATA #REQUIRED>

      <!ELEMENT uri (#PCDATA)>
        <!ELEMENT urgency EMPTY>
            <!ATTLIST urgency urgent (1 | 2 | 3)  #REQUIRED>
      <!ELEMENT docbase (#PCDATA)>
      <!ELEMENT expired EMPTY>
          <!ATTLIST expired expir (true | false) "false">
      <!ELEMENT summary (#PCDATA)>
```



# WebRACE System Architecture






# WebRACE Components

- Mini-crawler:
  - URL Queue & Dispatcher
  - URL Fetcher, Extractor & Normalizer
- Object Cache:
  - URL Indexer
  - Meta-information store
  - Content Store (supports versioning)
- Optimisations:
  - Multithreading
  - Proactive link filtering
  - Memory allocation & Garbage-collection



# WebRACE Implementation

**Java**

- Platform independence
- Strong typing
- Multi-threading
- Automatic memory management

**Mitsubishi Concordia Mobile Agents Platform**

- Java support
- Support for distributed operations and code mobility
- Persistence
- Messaging, event programming and coordination

**<XML>W3C eXtensible Markup Language (XML)**

- Self-descriptive format for communication between components (decoupling)
- Extensible and "open" grammars to specify services, user profiles, etc.
- Reusability of tools (Java classes for XML parsers)

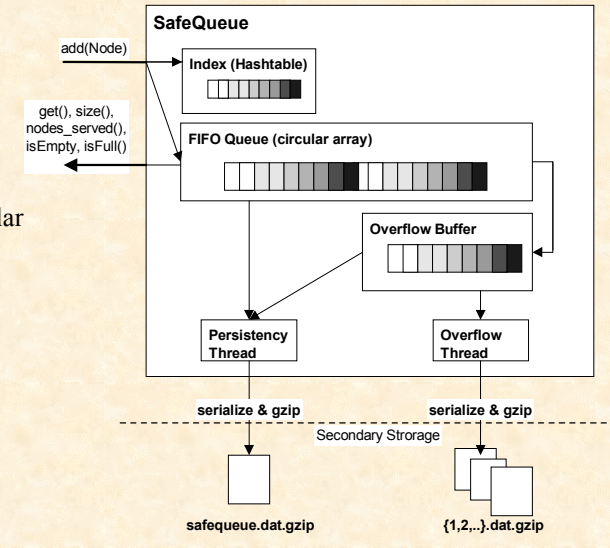
**Java Servlets**

- Interface programming
- Java support



# WebRACE Data Structures: SafeQueue

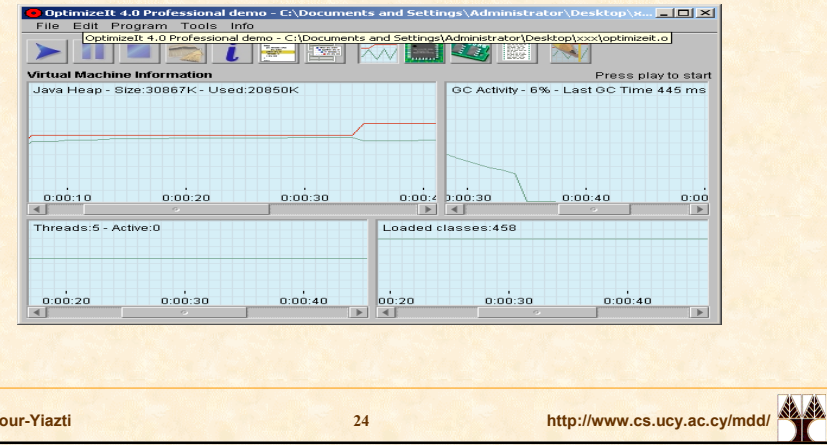
- ***SafeQueue*** is a "*thread-safe*" and "*persistent*" FIFO queue implemented in JAVA.
- **SQ** is implemented as a circular array of ***QueueNodes***.
- ***QueueNodes*** are any type of JAVA Object.



## Experimentation with SafeQueue

- In our experiments, we have configured the SafeQueue size to 2M nodes, i.e., 2M URLs. This number corresponds to approximately 27MB of Java Heap Space.
- Larger SQ's can be employed at the expense of heap size available for other components of our system.
- SQ is a data structure used by many parts of our System. We have used SQ for the implementation of WebRACE's Crawler (to enqueue requests for pages to be downloaded), and for WebRACE's Annotation Engine to enqueue "pages to be processed" requests.
- We are currently investigating ways to handle larger SafeQueue sizes by distributing the queue to different computing nodes.





## Fine-Tuning Performance

- We are using *Intuitive Systems' OptimizeIt* for measuring various performance properties of both our Crawler and the Filtering Processor Engine.

## Current Status & Future Work

- FIGI: A first prototype of eRACE tailored for gathering financial information from the Web
- 1st prototype of eRACE: agent-proxies for HTTP, NNTP, and POP3.
- WebRACE implementation & optimizations
- Distributed Crawler & Filtering Processor
- Near future:
  - Development of information ripping applications from the Web.
  - Using WebRACE to generate dynamically & publish WML content.