Intermediary Systems: A Survey

"Intermediary Infrastructures for the WWW," M. Dikaiakos, Computer Networks 45 (2004), 421-447

Outline

- Introduction.
- End-to-end arguments.
- Extending the edges.
- Proxies and CDNs.
- Intermediaries beyond proxies.

Definition

- *Software entities that intervene in the client-server exchanges taking place at the application layer of the Internet.*
- Deployed on hosts of the wireline and wireless network, along the route taken by client requests and server replies through the network (content path).
- From simple relaying and caching to more complex transformations involving the modification of requests and replies, and the creation of content.
- Extend the functionality and application performance offered by the network to its end-users, without violating the end-to-end principles of Internet design.
- Provide a "reusable and expandable set of services and functions, commonly needed by many applications to function well in a networked environment" [Middleware].

Motivation

- Intermediaries are important:
  - A useful abstraction for designing and studying emerging software infrastructures for the Internet services.
  - Will permeate the Internet because of the increasing demand for scalability, high-availability, personalization, localization, and ubiquity.

- Our goals:
  - Provide a survey of a wide range of intermediary systems and identify common characteristics and functional properties.
  - Examine the requirements and identify key components of intermediary systems.
  - Define a framework for comparing, modeling, and designing intermediaries.

“End-to-end” arguments in systems design

- Application-level functions usually cannot, and preferably should not, be built into the lower levels of a system.
- Functionality can completely and correctly be implemented only with the knowledge and the help of the applications standing at the endpoints of the communication system. Providing that functionality as a feature of communication systems is not possible [Saltzer et al., *ACM Trans. Comput. Syst.,* Vol. 2, No. 4, 1984]

Benefits of “End-to-end” arguments

- Flexibility
  - Can easily connect together computers with unknown applications.
  - Easy application upgrading at the end-point.
- Better performance
  - Lower-level system is common to all applications and it may not have as much information as the higher level.
- Lower cost and complexity in core of network.
  - Simplified network design.
- Application delivery guarantee
  - The communication layer can only guarantee delivery to the host.
- FIFO message delivery guarantee
- User empowerment.
  - Can run whatever application.

Example: Delivery guarantee

- The ack message in ARPANET was never found to be helpful to applications using ARPANET, why?
- Because knowing for sure that message was delivered to the target host is not very important.
- What the application wants to know is whether or not the target host has acted on the message!
- All manner of disaster might have struck after message delivery but before completion of the action requested by the message.
- The acknowledgement that is really desired is an end-to-end one, which can only by the target application—"I did it", or "I didn't".

Core vs. e2e

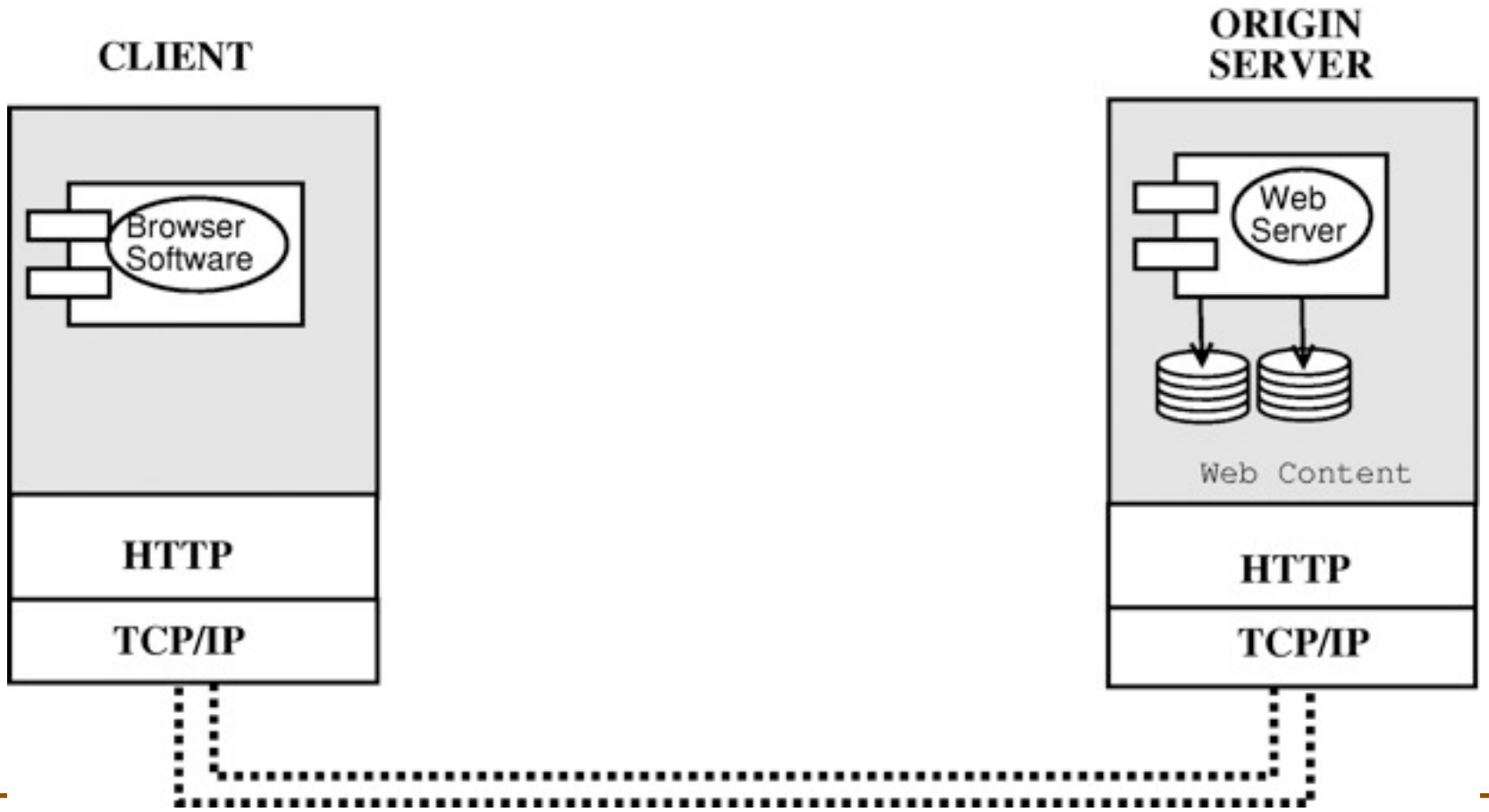
- The end-to-end argument is not an absolute rule but rather a guideline.
- Application layer has more information about the data and the semantic of the service.
- A lower layer subsystem has more information about constraints in data transmission (e.g., packet size, error rate)
- A new functionality at a lower level should have minimum performance impact on the applications that do not use the functionality.
- Lower layer implementation improves performance of a large number of applications

Tradeoffs

- Using the e2e arguments sometimes requires a subtlety of analysis of application requirements.
  - Example: if low levels of a telephone system try to accomplish bit-perfect communication, they will probably introduce uncontrolled delays in packet delivery. Such delays are disruptive to voice apps. It is better off to accept the damaged data and the participant to say “excuse me?”.

“End-to-end” principles and the Web

- The system architecture of the World-Wide Web is based on end-to-end principles: functionality and complexity are put at Web clients and Web servers.



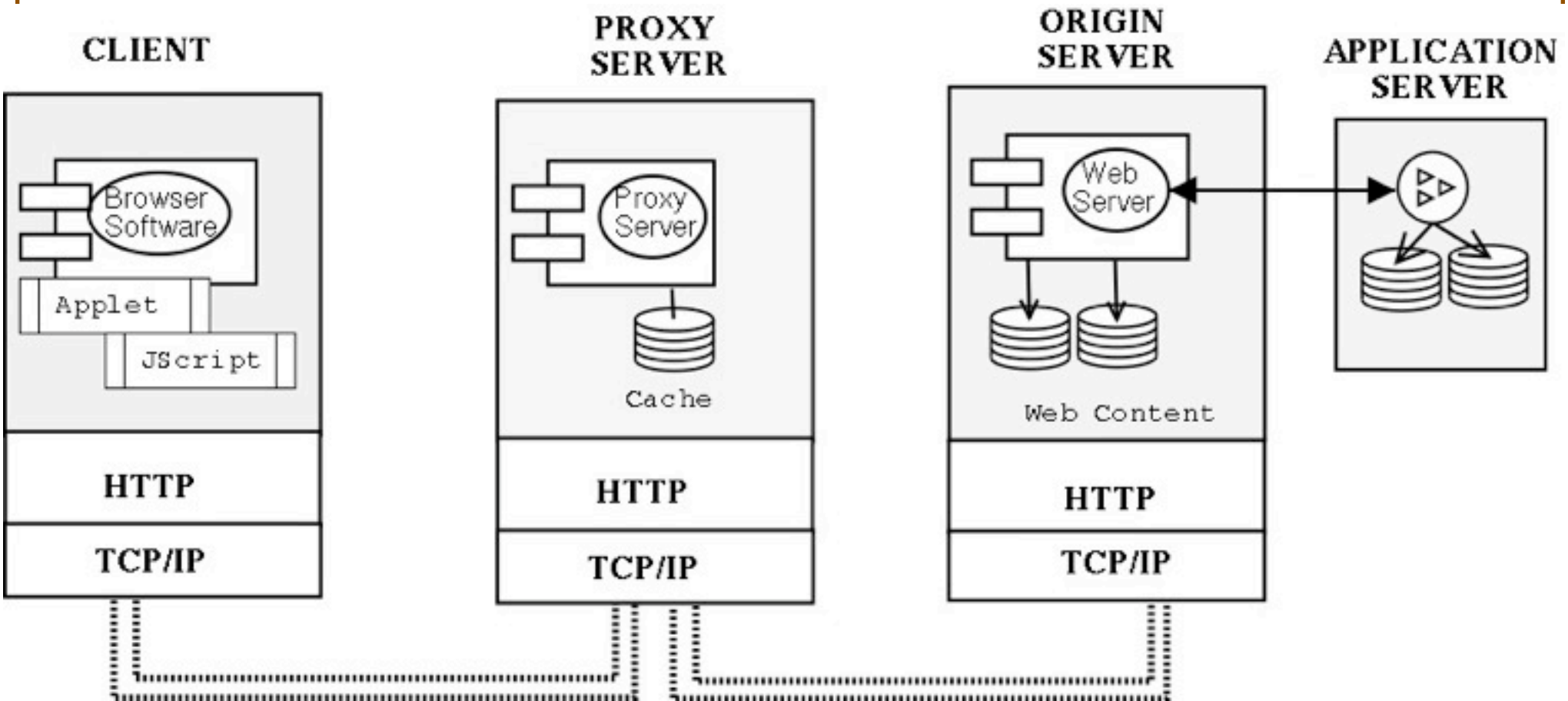
Departing from e2e

- Excessive loads of Web traffic: large latencies, overloaded Web servers.
- Wireless connectivity and mobile services: disconnected operation, content and service adaptation.
- More demanding applications: streaming media with stringent bandwidth requirements.
- Need for service differentiation: added-value, different service-levels, personalization, context and location awareness.
- Less sophisticated users: delegating configuration, protection, and control to a common point, part of the application execution context.
- Diversity of terminal devices: Support for wireless, mobility and ubiquity.
- Operation in an untrustworthy world.

Departing from e2e (ctd)

- Third party intervention
  - ISPs want to sell services, add value, and make money.
  - More complex role for commercial ISPs
    - vertical integration of transport/QoS with applications
    - control of what applications users can use.
    - filtering of unacceptable behavior.
- Need to control traffic:
  - Firewalls, Traffic filters, Network address translation elements
- Need for anonymity and accountability.

Extending the edges



Dynamic content management

- Web pages as collections of:
  - Dynamic information fragments extracted from back-end applications and databases (WebSphere).
  - XML fragments derived through queries to back-end databases (Weave).
- High-level abstractions for Web-site structure:
  - Object Dependence Graphs persistent data-structures (IBM's WebSphere).
  - OO Declarative programs (WeaveL language, INRIA's Weave project).
- Run-time policies for update propagation and consistency with back-end systems specified via:
  - Trigger monitors integrated with ODG's (WebSphere)
  - Declarative specification languages (WeavePRL).
- Run-time adaptability - flexibility to choose at run-time which codes to invoke is provided by the Accessible Business Rules framework which contain run-time decision points (WebSphere).

Efficient Creation of Dynamic Content

- *Source: "A Publishing System for Efficiently Creating Dynamic Web Content," J. Challenger et al., INFOCOM 2000.*
- A system employed for the Olympic Games 2000 Web sites.
- Problems with dynamic content:
  - A typical dynamic page may require several orders of magnitude more CPU time to serve than a typical static page of comparable size.
  - Correctly and consistently updating dynamic pages whenever there are changes in underlying data (a change in underlying data may affect multiple pages).
  - Invalidating dynamic pages cached in main memory or file systems, whenever there are updates in the pages' content:
    - which cached pages are affected by updates in underlying data?

Managing Dynamic Content (ctd)

- Dynamic pages are implicitly updated any time an embedded fragment changes.
- Consistency becomes an issue with the fragment-based approach when the pages are being published to a cache or file system.
- The fragment-based approach for generating Web pages:
  - Makes it easier to design Web sites with a common look and feel.
  - Facilitates the embedding of common information into several Web pages.
  - Enhances the management of sets of Web pages which contain similar information.

Managing Dynamic Content (ctd)

- Composition of dynamic pages out of fragments:
  - Fragments represent parts of Web pages which change together.
  - When a change to underlying data occurs which affects several Web pages, the fragments affected by the change can easily be identified.
  - It is possible for a fragment to recursively embed another fragment: fragments are either **atomic** or **complex**.
- The overhead of composing a page from simpler fragments is usually minor if compared to the overhead of constructing the page from scratch.
- Using the fragment approach, it is possible to achieve significant performance improvement without caching dynamic pages and dealing with the difficulties of keeping caches consistent.
- It is possible, however, to cache dynamic pages - this capability is integrated with the fragment management system.

Elements of the IBM approach (WebSphere)

- A system for authoring Web pages as collections of fragments: employ templates written in a markup language.
- A system for extracting inclusion relationships between fragments and pages: parsing templates and constructing an Object Dependence graph (ODG).
- A system for managing the propagation of changes in underlying data to the pages of the Web site: employing graph traversal algorithms on the ODG to determine propagation steps.
  - ODG incorporated into the Accessible Business Rules framework (ABR), which provides developers with access to business rules
- Accessible Business Rules (ABR) middleware: enables application development where time and situation-variable parts are determined by **business rules**.

Business Rules [Wikipedia]

- A **Business rule** is a statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behavior of the business. Business rules:
  - describe the operations, definitions and constraints that apply to an organization.
  - can apply to people, processes, corporate behavior and computing systems in an organization, and are put in place to help the organization achieve its goals.
- A **business rules engine** is a software system that executes one or more business rules in a runtime production environment.
  - The rules might come from legal regulation, company policy ("All customers that spend more than $100 at one time will receive a 10% discount"), or other sources.
  - A business rule system enables these company policies and other operational decisions to be defined, tested, executed and maintained separately from application code.
  - Rule engine software is commonly provided as a component of a business rule management system which, among other functions, provides the ability to: *register, define, classify, and manage all the rules, verify consistency of rules definitions*, *define the relationships between different rules, and relate some of these rules to IT applications that are affected or need to enforce one or more of the rules*.

WebSphere's Accessible Business Rules

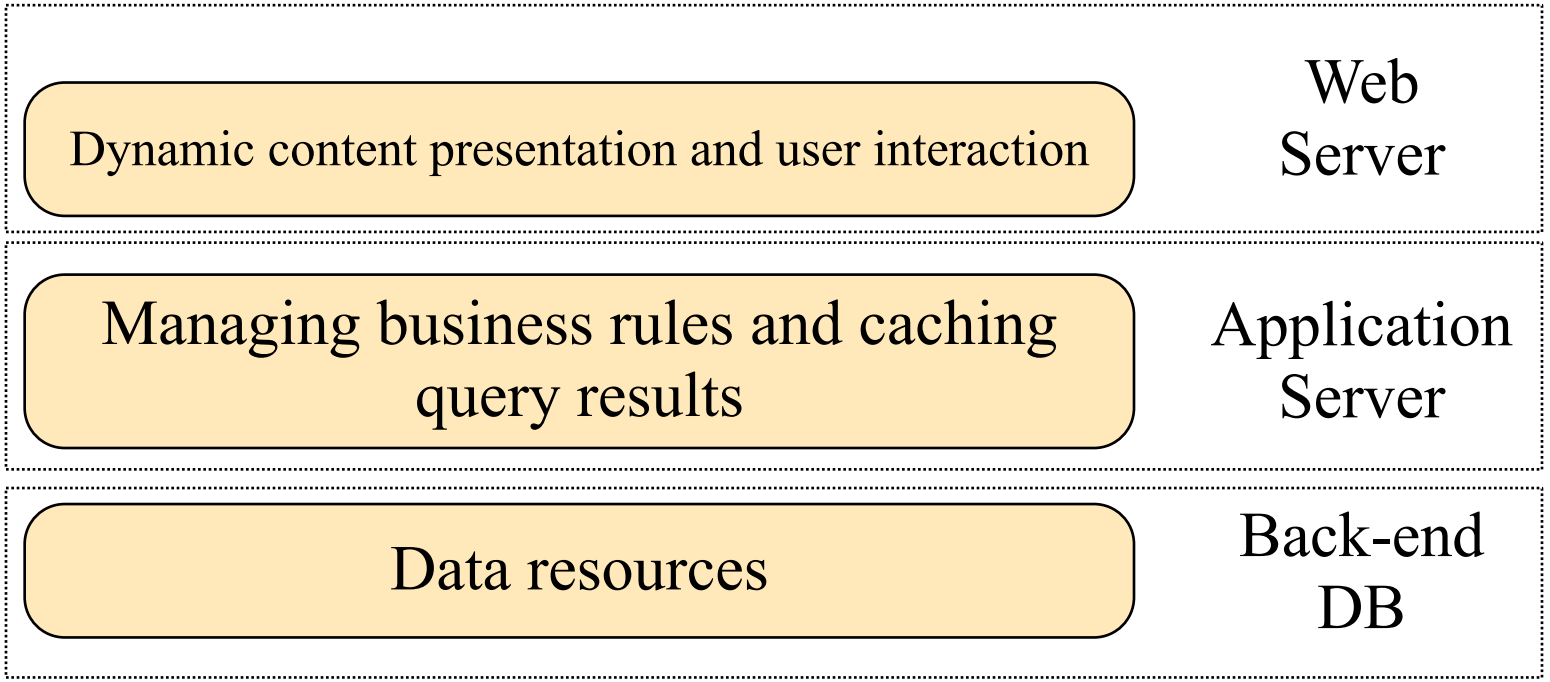
- Applications contain decision points, which query an ABR Server at run-time and retrieve the business rules that apply under a given context.
- Once retrieved, these rules are "fired," adapting the behavior of the application at the particular decision point.
- Business rules are managed externally to the application.

```
Q1:
        SELECT * FROM RULEUSETABLE WHERE
                CONTEXTID               LIKE    'customerLevel'
        AND     TYPE                    LIKE     'classifier'
        AND     COMPLETIONSTATUS        LIKE     'ready'
```

```
Q2(userClassification):
        SELECT * FROM RULEUSETABLE WHERE
                CONTEXTID               LIKE    'promotion'
        AND     CLASSIFICATION          LIKE     $1
        AND     TYPE                    LIKE     'situational'
        AND     COMPLETIONSTATUS        LIKE     'ready'
```

IBM WebSphere: General Purpose Cache

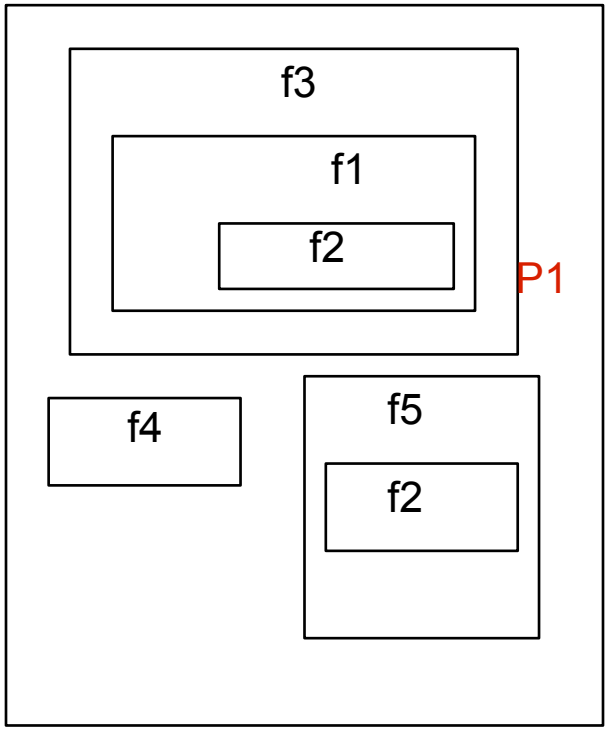
- ABR framework architecture: three tiers

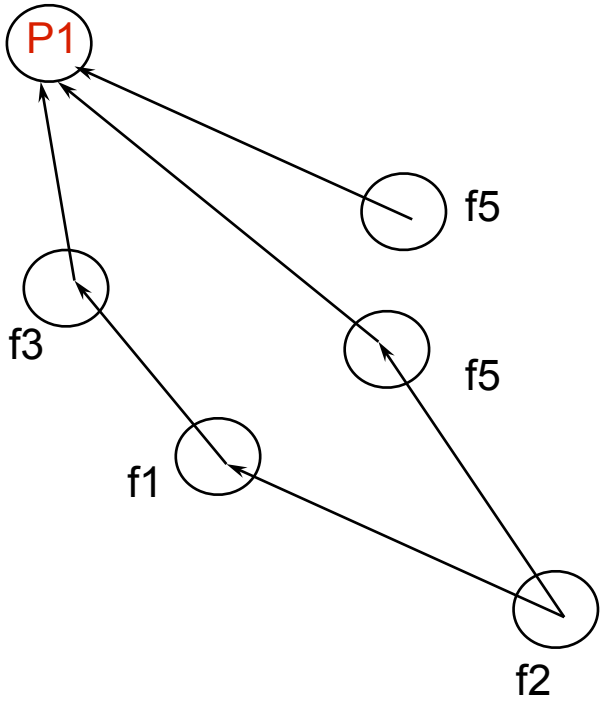


IBM WebSphere: General Purpose Cache

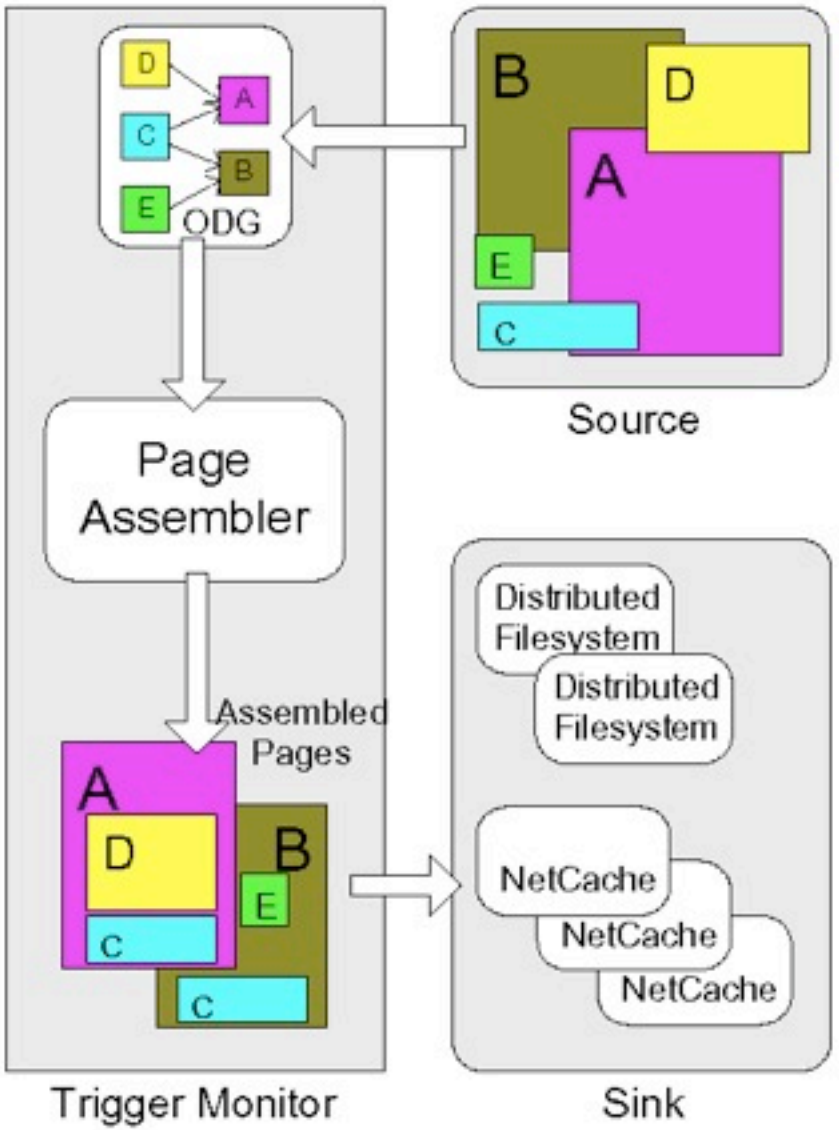
- In the Accessible Business Rules (ABR) middleware:
  - A cache reduces the number of queries to remote databases by storing query results.
- General-Purpose Software (GPS cache) cache :
  - efficient code for storing data in memory, on disk, or both.
  - optimized support for invalidating objects based on expiration times and for logging cache transactions.
- The GPS cache in ABR, stores results of queries ultimately made to a database.
  - Problem: how to keep the cache current after database updates. Determine which queries are affected by changes that occur to the database.
- Solution: Data Update Propagation (DUP).
  - Dependency relationships between attributes and query are represented by an object dependence graph (ODG).
  - DUP Value-aware update policy :
    - When attributes change, consider old and new attributes values to determine how to update cache.
    - Automatically generate ODG's from the queries within an ABR application.

Web-page fragments and ODG's





Publish Process



- Trigger monitor: Takes objects from sources, constructs pages and writes them to sinks.

INRIA Weave

Source: K. Yagoub, D. Florescu, P. Valduriez, V. Issarny, "Caching Strategies for Data-Intensive Web Sites" , Proc. of the Int. Conf. on Very Large Data Bases (VLDB) ,Cairo, Egypt, 10-14 September , 2000

- Main goal: separate three key concerns of Web-site management:
  - Web site structure and content specification
  - Web-page presentation and graphical style
  - Implementation: content assembly and HTML creation

INRIA Weave

- Caching data-intensive Web sites
  - Content is dynamically extracted from a database.
  - Relies on the declarative specification of web sites
  - Specification of website is distinguished from its implementation. The mapping between raw data and logical model of web is described by declarative language (WeaveL)
  - Specification of website structure and content is separated from the graphical representation of the pages.
- Weave: tools for design, implementation, profiling, deployment and monitoring of site.

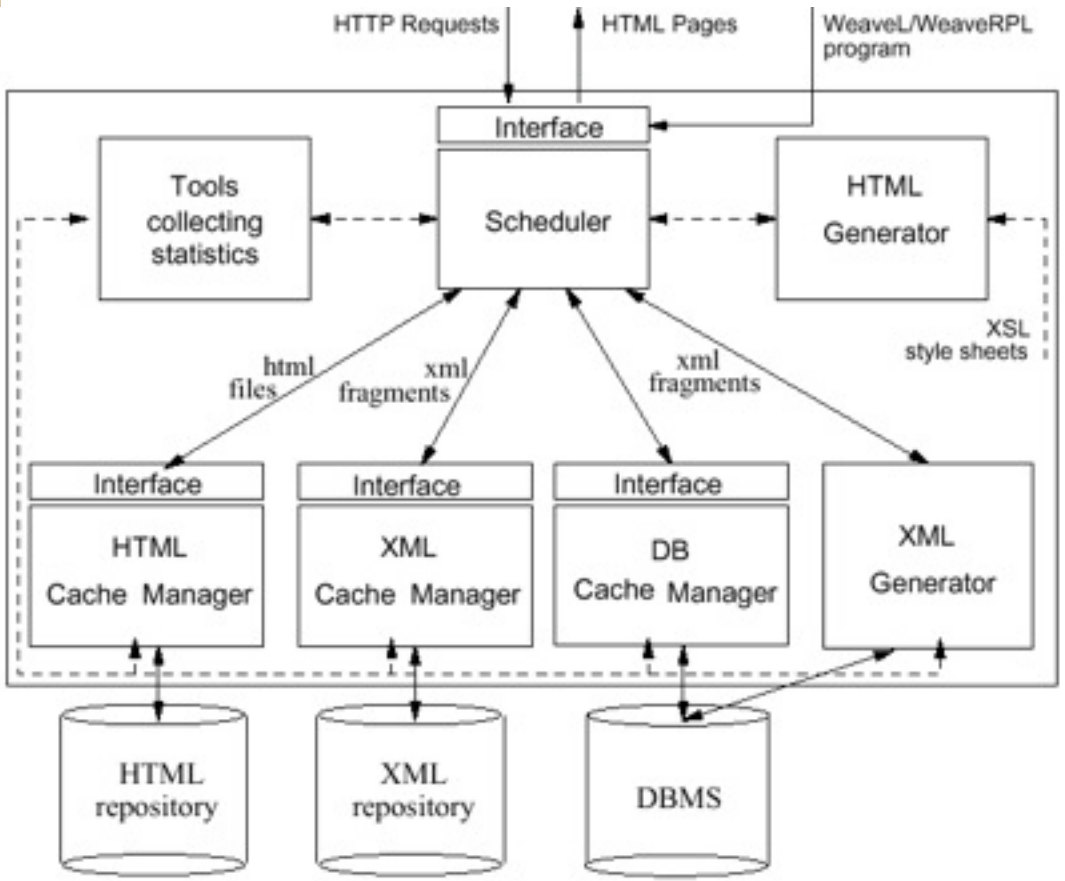
INRIA Weave (ctd)

- Website *declarative specification*:
  - WeaveL program: XML graph data model to describe *structure* and *content*. An instance for a site is called XML site graph.
  - XSLT to describe graphical representation.
- WeaveL program: Consists of a site class specification modeling a collection of homogeneous pages in a website. Each page is an *instance* of a class. The specification includes:
  - declaration of parameters identifying an instance
  - SQL query that gives all the instances of the class
  - the specification of the data contained in an instance of the class
  - the db queries that have to be evaluated to build the correct links between the pages
  - the specifications of the forms embedded in the page.
- A page request specifies the class of the file and values for zero or more parameters.

INRIA Weave (ctd)

- Data Materialization: re-using intermediate results of various computations to answer subsequent queries.
  - **What** kind of data must be materialized?
    - Results of db queries, XML fragments and HTML files.
  - **When** must materialization must be performed?
    - (i) proactively, (ii) upon request, (iii) prefetching
  - **Where** should the materialized intermediate result be placed for effective performance improvements?
    - (i)database server, (ii)web server, (iii) proxy, (iv) web client.
  - **How** are updates propagated to the materialized data?
    - (i)push, (ii)pull.
  - **Which** data must be materialized or computed upon request?
    - Expensive to compute, do not require frequent propagation, and requested frequently.

Architecture of Weave



- Scheduler: Interprets the runtime policy and coordinates the behavior of the other components.
- Cache manager: Retrieves components with data related to the page. Enforces policies.
- Repositories: Store the data. Multiple caches for scalability.
- XML Generator: Issues queries to the DBMS and produces XML fragments.
- HTML Generator: Generates HTML pages from fragments and XSLT programs
- Statistics Manager: In charge of storing and summarizing data about runtime behavior.

INRIA Weave (ctd): WeaveL

```
define class CustomerNat ($NK)
{instances using Q0 }
{
 data  nation_name using Q1 ;
 link customer to Customer($CK) using Q2 ;
}
define query Q0 as select nationkey as $NK from Nation;
define query Q1 as select name as nation_name
                   from Nation where nationkey=$NK;
define query Q2 as select custkey as $CK, name as anchor
                   from Customer where nationkey=$NK;
```

INRIA Weave (ctd): XML Fragment

```
<XML_fragment id="CustomerNat_6">
    <class> CustomerNat </class>
    <parameter> 6 </parameter>

    <data_fragment name="nation_name">
        <data_value> France </data_value>
    <data_fragment>

    <link_fragment name="customer">
        <link_item>
            <XML_fragment id="Customer_402">
                <class> Customer </class>
                <parameter> 402 </parameter>
            </XML_fragment>
            <anchor> Customer#000000402 </anchor>
        </link_item>
        ....
    </link_fragment>
</XML_fragment>
```

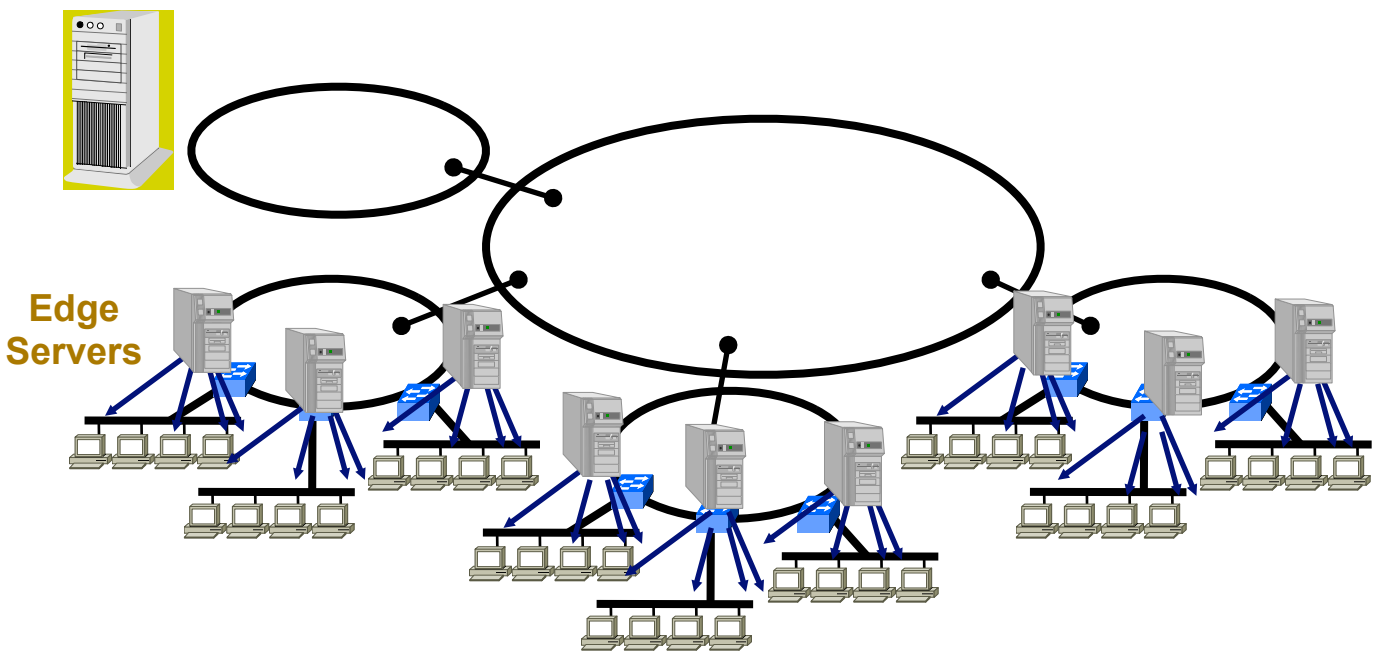
INRIA Weave (ctd): WeaveRPL

- The data materialization process is driven by a declarative specification of runtime and caching policies described in WeaveRPL.

```
Cache XML:
{   /* Container definition */
     define container contXML as
        select Supplier(SK):fragments{name, customer}
        where SK < 100;
     /* Global constraints */
          frequency > 0.2 and size < 100 KB;
     /* ECA rules */
          onInit compute all;
          onTimer(5mn) reinit;
          onFull(50MB) apply LRU;
}
```

Proxies and Content-Distribution Networks

- Proxy caches reduce response times, server loads, HTTP traffic.
- Organized as hierarchical and co-operative caching infrastructures.
- Content Distribution Networks: belong to one administrative entity and comprise geographically distributed collections of cooperating edge proxies.



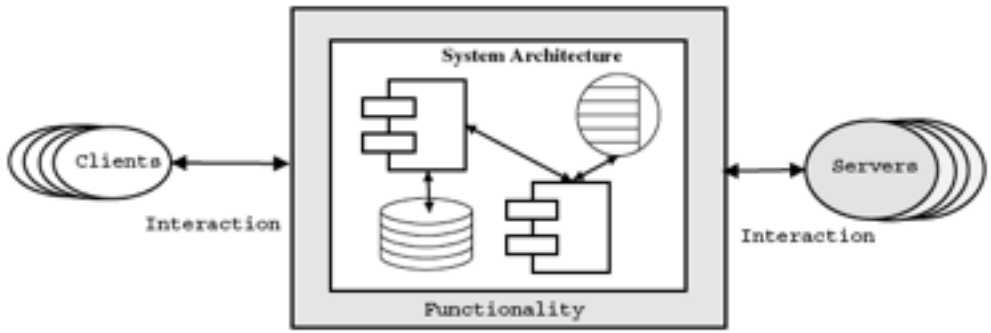
Content-Distribution Networks (CDN)

- The purpose of a CDN is to proactively replicate the content of one or more related Web sites to strategically chosen locations across the Internet, then transparently redirect browsers to the nearest or most responsive cache.
- *Getting content to the edge* to improve service availability and reduce latency.
- Performance tradeoffs with caching hierarchies
  - Reduced server load, reduced network traffic and reduced end-user latency

    but
  - Inter-cache communication overhead, delays incurred at each level of the hierarchy and performance bottlenecks at higher-level proxies
  - (P.Rodriguez, C. Spanner, E.Biersack. Web caching architectures: Hierarchical and distributed caching. In *Proceedings of the 4th Web Caching Workshop,* 37–48, San Diego, CA,1999.)

Beyond Web proxies

- We examine intermediaries that extend the paradigm of typical Web intermediaries along three dimensions:
  1. The functionality provided by an intermediary beyond proxying and content caching: capabilities, services, policies.
  2. The system architecture of an intermediary, describing its composition in terms of components, the division of roles and functions between them, their inter-relationships, their placement across the network and their ownership.
  3. The interaction between intermediaries and their counterparts, described in terms of communication patterns, supported protocols, access models.



Functionality

- Customization: restructuring the presentation of content according to user-preferences, terminal-device capabilities, context of use, location, etc.
- Filtering: semantic analysis of content retrieved from origin servers; pertinent to intelligent techniques for service personalization and localization, protection from viruses, indecent content, etc.
- Annotation: generation and dissemination of additional information and meta-information (summaries, keywords, highlights, ad banners).
- Transcoding: transformation from one format to another, to enable presentation in different terminals and to optimize transportation across wireless links.
- Protocol Translation between wireless and wireline protocols.
- Content creation: produced from application code off-loaded, cached and executed on intermediaries, or from the aggregation of content retrieved from multiple origin servers.

System Architecture

- Structure: centralized/tightly integrated versus distributed intermediaries.
- Deployment: near origin server, near client, in the network, in the intranet.
- Ownership: client device, ISP, intermediary service provider, CDN, content provider, enterprise intranet, etc.
- Complexity: special purpose, “light” components vs. components with complex functionality (caching, versioning, indexing, profiling).
- Support for configurability and programmability:
  - Configuration parameters: determine the set of operations to be invoked in a given context; hard-wired or extracted from meta-data files
  - Generic execution environments: supporting the dynamic off-loading of intermediary entities into the infrastructure (JVM, Mobile agents, Jini); provide APIs, libraries, patterns for programming new services.
  - Compositional frameworks: providing components used as building blocks for defining new services according to higher-level programming models.

Interaction

- Communication mode: synchronous (on-demand) versus asynchronous.
- Access model: push (server-initiated) versus pull (client-initiated).
- Protocol support: HTTP, stripped-down HTTP, SMTP, NNTP, WAP.
- Supported media: wireline versus wireless.

Classifying Intermediaries

- We examine three categories of intermediary systems (based on their functionality and focus):
  - Notification intermediaries (aka information-dissemination systems): driven by end-user profiles, operate asynchronously, even when an end-user is disconnected.
  - Intermediaries for wireless and mobile services.
  - Intermediary infrastructures: overlay networks extending the support of the core network for the development and deployment of new services.

Notification Intermediaries

- Monitor content in origin Web servers on behalf of subscribed users.
- Evaluate the relevance of changes with respect to stored user profiles (long-term, continuously evaluated queries).
- Notify subscribers accordingly (push or pull).
- Provide support for filtering, annotation and aggregation of content.
- Examples of notification Systems:
  - SIFT (Stanford Univ.) - for Usenet News; information-retrieval style profiles
  - AIDE (AT&T Research) - for the Web; emphasis on versioning and differencing of Web pages.
  - Grand Central Station (IBM) - for the Web, Usenet news, email; boolean-structured predicated language for profile specification; combined with Web casting dissemination.
  - FIGI (Univ. Cyprus) - for the Web, Usenet news, email; XML profiles; mobile-agent based architecture, dissemination with mobile agents, email, and SMS.

Stanford Information Filtering Tool (SIFT SIFT)

- Geared towards efficient large-scale information dissemination services to users that subscribe their interests to SIFT servers.
- The user submits to the system profiles - one per topic of interest
- Profile includes :
  - Query : Boolean or Vector Space Model
  - Frequency of notification
  - Amount of info to receive
  - Life of profile
  - Email address, profile identifier
- Implementation:
  - Collect new documents daily from USENET News
  - Match each article against stored queries
  - Generate notifications - daily and non-daily
  - Sort files by (user, profile) and mail document to users

AT&T Internet Difference Engine (AIDE)

- Tool for tracking and viewing modifications to World-Wide-Web resources.
- It is comprised of a centralized notification server, a version archive and a difference engine that identifies and displays changes in webpage content.
- Users subscribe a list of URLs and configuration parameters.
- AIDE supports recursive tracking and differencing of pages.
- Changes to an HTML are presented with highlighting.

Grand Central Station - IBM

- Designed to find and summarize information on the Web as soon as it becomes available.
- GCS pushes the results to a variety of devices from a desktop to Personal Digital Assistants (PDA).
- The task of collecting and sorting through vast amounts of information is split to several geographically-divided GCS Gatherers.
- Crawlers search out information from both readily available and obscure data sources, Web servers, news servers, database systems and file systems
- Data is then organized by a Recognizer according to type -- graphics, database files, Web documents, e-mails or sounds.
- Subsequently the Selector filters the information to remove irrelevant material before handing it off to the Summarizer.

Grand Central Station – IBM (ctd)

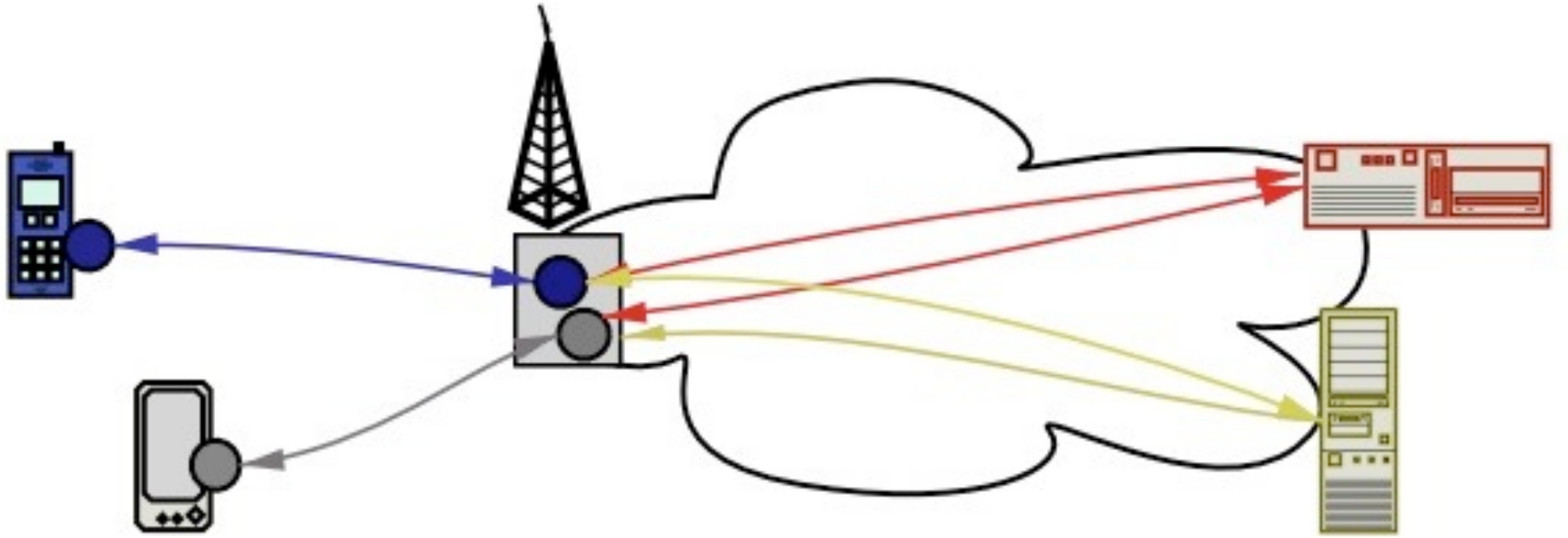
- The Summarizer is a collection of plug-in programs distinguished by data type; it produces a summary represented in a metadata format known as the SOIF (Summary Object Interchange Format).
- A Web server associated with each Gatherer makes the SOIFs available to a central component called the Collector. From the SOIFs, the Collector creates a database that is essentially a map of the digital universe.
- The Profile Engine matches this information to the interests and needs of users, starting with the user's queries, it constructs information profiles that it constantly matches against the incoming information.
- When new information is discovered, it distributes them to Administration Servers that deliver them to the client's desktop machine or PDA.

Intermediaries for Mobility & Wireless: Requirements

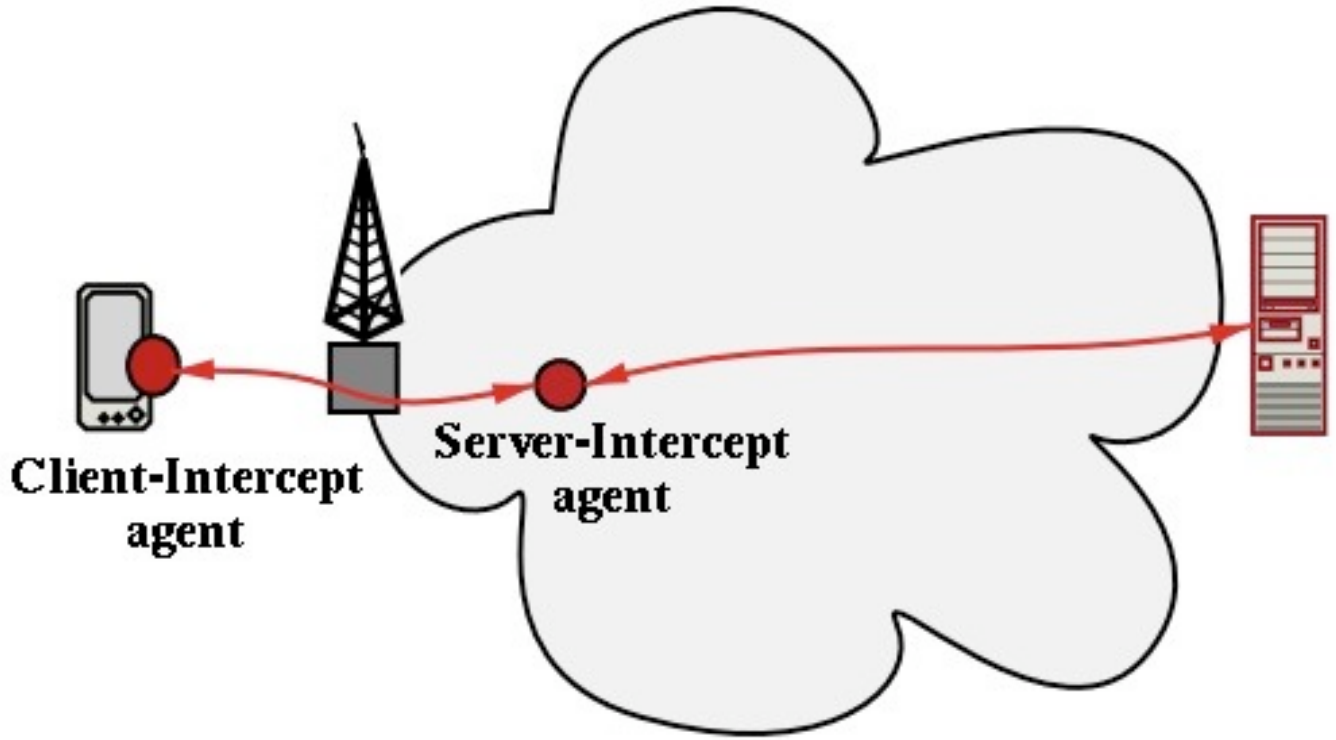
- Optimize client-server communication over the wireless medium.
- Support personal, physical, and service mobility.
- Support both synchronous and asynchronous interaction, coping with frequent disconnections.
- Support seamless access from a variety of devices.
- Adapt content to terminals of different type.
- Support the provision of content in multiple formats (HTML, WML, XML) to the same device over the same link.
- Adapt content to situational and personal preferences (personalization, localization, context-awareness).
- Guarantee high availability, robustness, and performance scalability.

Distributed agent-based approaches

- Deploy special-purpose, “light” components (“agents”) on the wireline network and/or the mobile hosts.
- These agents collaboratively mediate the communication between origin servers and mobile terminals.



IBM's WebExpress

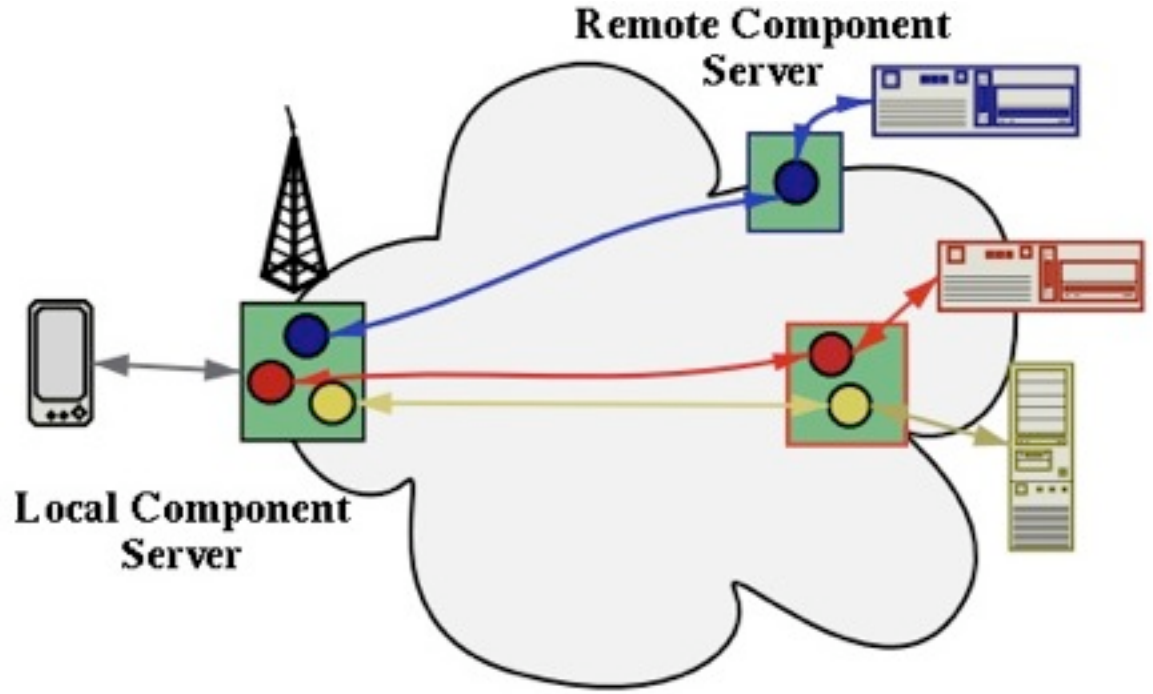


- By IBM [Housel, Samaras, Linguist, 1998].
- Supports wireless WWW access.
- Provides HTTP optimization over wireless channel, header caching, disconnection management.

IBM's WebExpress (ctd)

- **Client/intercept/server model**: a client-side intercept agent (CSI) and a server-side intercept agent (SSI) inserted into the data path between the client and the server.
- Only change: Specify at the browser the (local) IP address of the CSI as the browser's proxy.
- The CSI communicates with an SSI process using a reduced version of HTTP. The SSI reconstitutes the HTML stream and forwards it to the designated proxy server.
- **Caching**: At both the CSI and the SSI.
  - The SSI cache is populated by responses from the requested web servers.
  - If requested object is in CSI's cache, it is returned; otherwise, the cache at the SSI is checked. When a cached object is referenced, the CSI checks whether the coherency interval has been exceeded.
  - The client cache replacement policy is an LRU (least recently used) policy .
- **Differencing** : HTML streams often contain a lot of unchanging data. A common base object is cached at both the CSI and SSI. SSI computes difference between base object and response and transmits the difference. The CSI merges the difference with its base object.
- **Header Reduction** : The CSI allows HTTP header to flow in the 1st request and only the new information is sent subsequently.
- **Disconnected Operation**: The client relies solely on CSI cache. "Asynchronous-disconnected" mode: requests are queued and resumed when connectivity is re-established.

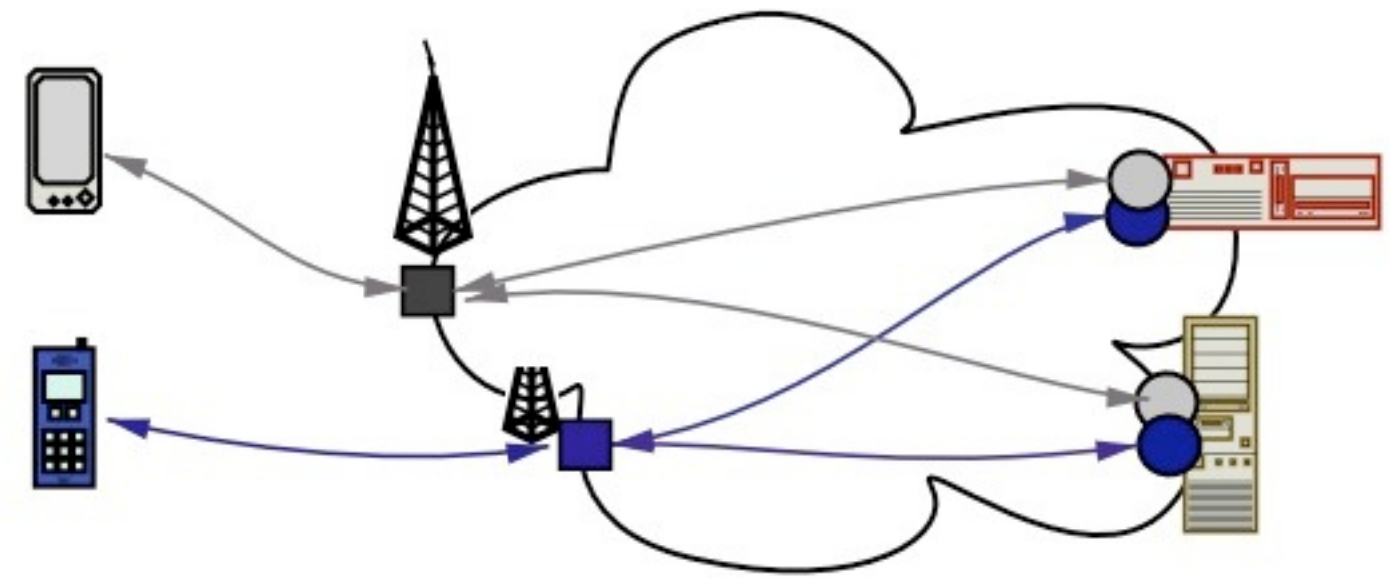
Web Stream Customizers



- Steinberg & Pasquale (WWW 2003). Provides redirection of HTTP traffic, protocol and content adaptation, compression and caching.
- Local Component Server attached to client; hosts multiple local components (one per origin server).
- Remote Component Server attached to origin server; hosts multiple remote components (one per client).

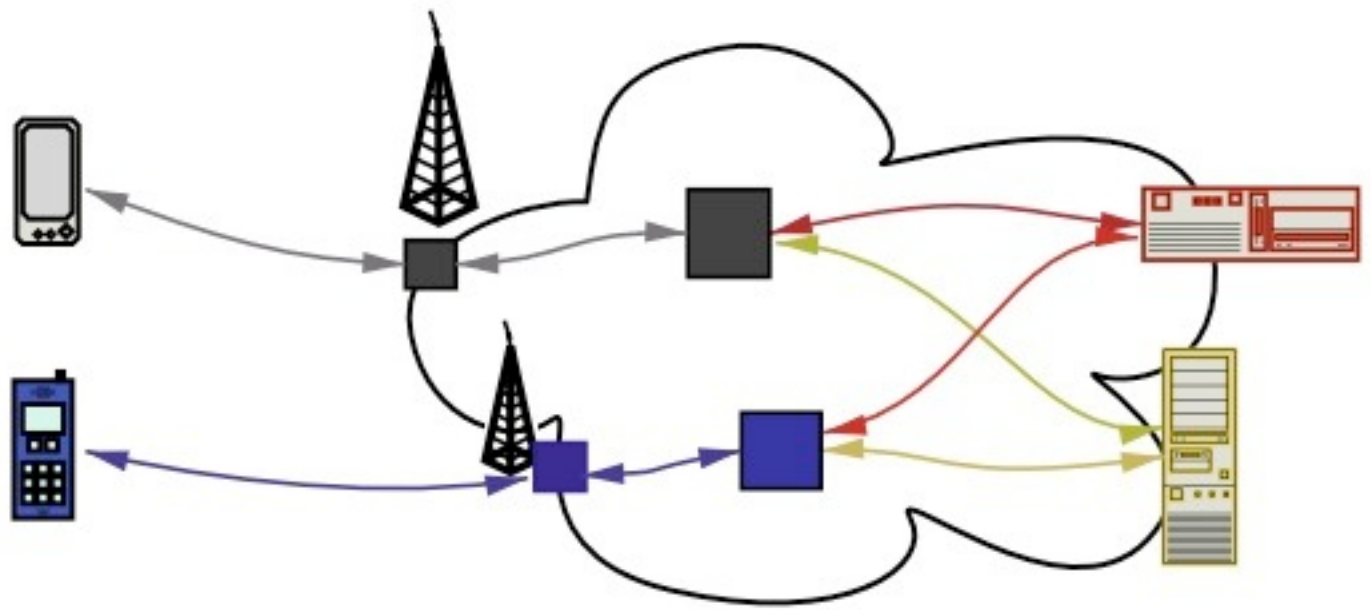
End-to-end solutions

- Origin servers store content in multiple formats compatible with various wireless clients or adapt it on-the-fly according to client, connection, or end-user profiles.
- However: adapted content or adaptation software has to be inserted at each origin server (and updated as new terminals and protocols become available).



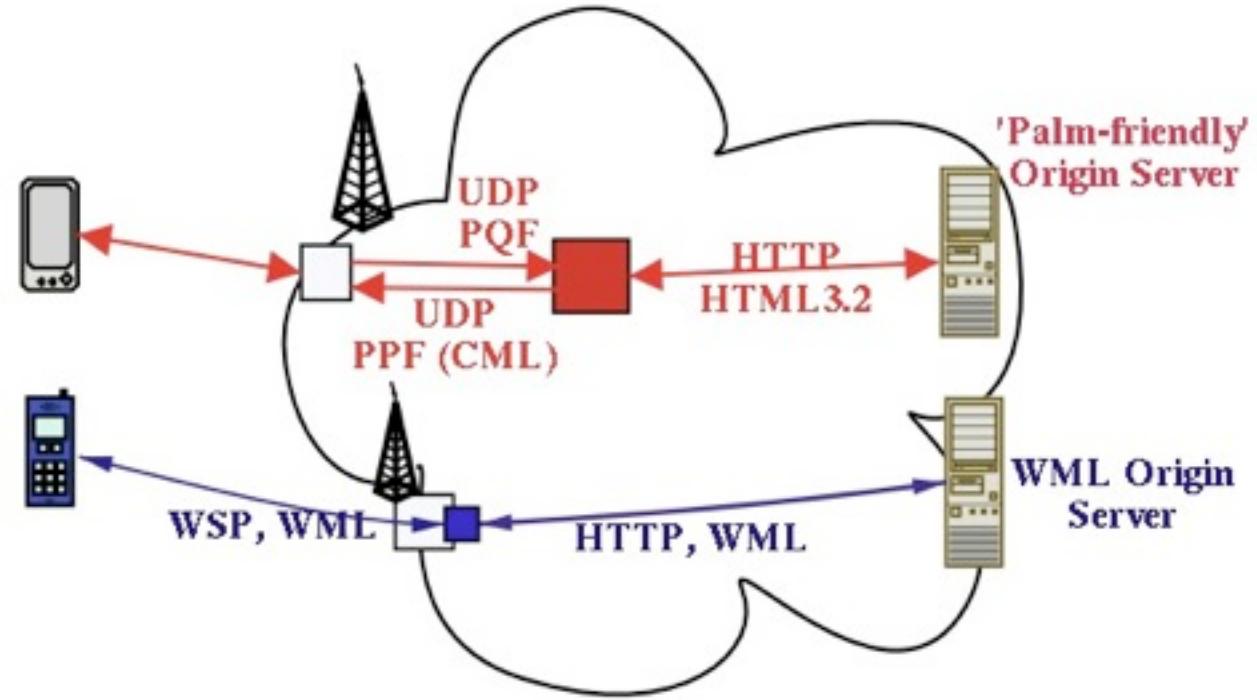
Proxy-based solutions

- Deploy powerful adaptation intermediaries in the net, to act as delegates for a specific family of devices.
- Can be combined with end-to-end solutions.



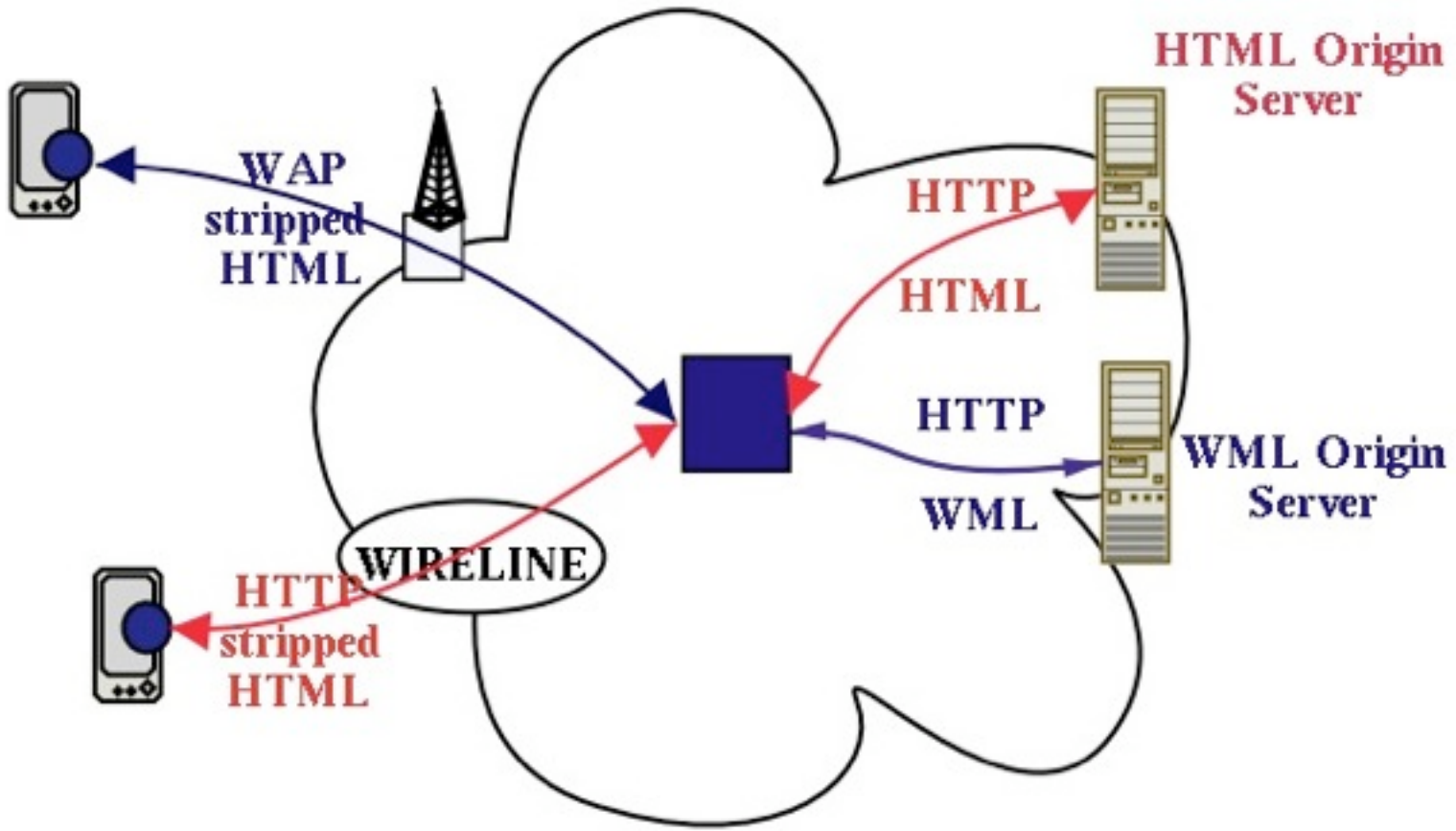
WAP Gateways and Web Clippings

- Combine the end-to-end approach (“specialized” ends) with the proxy-based solution (proxy providing optimization of communication).
- Provide translation between HTTP and wireless protocols, and lower-level optimizations for the provision of Web access over wireless devices.
- Presume the publishing of content into specific formats (WML and HTML3.2, respectively).



Handspring's Blazer

- Combination of a powerful adaptation proxy with a device-specific micro-browser.



Intermediary Infrastructures

- Intermediary solutions need to support, additionally:
  - Large numbers of simultaneous end-users.
  - Big heterogeneity of terminal devices.
  - High-throughput of requests for service.
  - Bursty workloads.
  - High-availability and incremental scalability in service provision.
  - Ease in the definition and deployment of new services.
- Requirements:
  - To shift computation, storage and complexity from centralized proxies, terminals, mobile-base stations, into the networking infrastructure.
  - Distributed, cooperating, network-centric components.
  - Enhance the programmability or configurability of intermediary components.

Intermediary Infrastructures

- Proposed approaches suggest a separation of concerns:
  - Content presentation.
  - Communication optimization.
  - Aggregation of content.
  - Orchestration of functions.
  - Robust execution.
- Solutions typically include some type of multi-tier architecture:
  - Lower tiers provide the execution environments of intermediary services, offering support for load-balancing, high-availability, etc.
  - Middle tiers provide mechanisms for specifying the orchestration of intermediary functions.
  - Upper tiers provide support for adapting the content to particular terminal devices and network connections.

Examples of Intermediary Infrastructures

- BARWAN project (UC/Berkeley): support service access from mobile clients roaming across heterogeneous wireless networks.
  - Lower tier: Scalable Network Services
  - Middle tier:TACC servers (transcoding, annotation, caching, customization)
  - Upper tier: handles presentation of data to various terminals.
  - Compositional framework: inspired by UNIX pipes.
  - Basic component: TACC workers,supporting basic functionalities (transformation, aggregation, customization, caching).
- Ninja architecture: robust, Java-based infrastructure for Internet services
  - Lower tier: Bases and vSpace execution environment.
  - Middle tier: Active Proxies provide transformational support between services and clients.
  - Upper tier: Units represent terminal devices in the infrastructure.
  - Compositional framework: introduces a path abstraction, comprising operators and connectors, put together according to four patters (wrap, pipeline, combine, replicate).

Examples of Intermediary Infrastructures

- WBI (Web Browser Intelligence, IBM Almaden): assembles complex intermediary systems from simpler components.
- Compositional framework:
  - Based on five building blocks: request editors [upper tier], generators [lower tier], document editors [middle tier], monitors and autonomous functions [logging and book-keeping functions].
  - These are composed into information streams, packaged as WBI plugins, and connecting origin servers with terminal devices.
  - Multiple WBI plugins can be composed into WBI services.
  - Composition of WBI plugins can be dynamic, using a rule-based approach.
- iMobile (ATT research):
  - iProxy: programmable proxy server designed to host agents and services in Java.
  - Programming abstractions: devlets, applets, infolets.

Bay Area Research Wireless Access Network (BARWAN) project (UC/Berkeley)

- Provide service to mobile clients roaming across heterogeneous wireless networks.
- *Key Concepts*:
  - Overlay networking: the unification of several heterogeneous networks, of varying coverage and performance, into a single logical network that provides the best of all the networks. Traditional cell-based (horizontal) roaming and transparent roaming across the constituent networks (vertical roaming) →TCP enhancements and corrective agents in the infrast.
  - Dynamic Adaptation: Deals with varying network conditions and client heterogeneity. Includes adaptive mechanisms for TCP, format conversion, real-time video transcoding, dynamic quality/ performance tradeoffs, and dynamic generation of customized UI for small devices.

BARWAN project (ctd)-Architecture

Service: Service-specific code
•Workers present human interface to what TACC modules do, including device-specific presentation •User interface to control the service •Complex services: implement directly on SNS layer
TACC: Transformation, Aggregation, Caching, Customization
•API for composition of stateless data transformation and content aggregation modules •Uniform caching of original, post-aggregation and post-transformation data •Transparent access to Customization database
SNS: Scalable Network Service support
•Incremental and absolute scalability •Worker load balancing and overflow management •Front-end availability, fault tolerance mechanisms •System monitoring and logging

BARWAN project (ctd)

- TACC: Compositional framework: inspired by UNIX pipes. Simplifies service creation and hides the details of the SNS layer.
- Basic component: TACC workers supporting:
  - **T**ransformation ("one-to-one"): changes on a single data object
  - **A**ggregation ("many-to-one"): Search engines, crawlers, "my headlines"
  - **C**aching: Both original and TACC-generated content
  - **C**ustomization : Per user: content generation, Per device: data delivery, content "packaging"

BARWAN project (ctd)

- Easy to write applications:
  - Rapid prototyping
    - Isolate workers from details
  - Easy to incorporate existing/legacy code
    - Few assumptions about code structure
    - Must support variety of languages
  - Composition to leverage existing code

Ninja Architecture (UC/Berkeley)

- Robust platform to enable dynamic service composition.
- Goals:
  - Enable the broad innovation of robust, scalable, distributed Internet services and
  - Permit the emerging class of extremely heterogeneous devices to seamlessly access these services [GWB00].
- Service:
  - *Software embedded in the Internet infrastructure that exports a network-accessible, typed, programmatic interface, and that provides strong operational guarantees.*
  - *It must exhibit scalability, availability, fault tolerance, data consistency and persistence.*

Ninja Architecture

- Ninja architecture: robust, Java-based infrastructure for Internet services
  - Lower tier: Bases and vSpace execution environment.
  - Middle tier: Active Proxies provide transformational support between services and clients.
  - Upper tier: Units represent terminal devices in the infrastructure.
  - Compositional framework: introduces a path abstraction, comprising operators and connectors, put together according to four patters (wrap, pipeline, combine, replicate).
  - Supports dynamic composition of horizontal services into a path, as well as adaptation along that path.
    - Adaptation is done by active proxies: they transform data types, adapt protocols or content.

Ninja Architecture (ctd)

- The composition process involves five entities:
  - Internet Service Providers: Resource providers for information requested by a query.
  - Network Service Providers: Provide connection to the various devices through varying type of networks.
  - Automatic Path Creation Service Provider (APC): Coordination of composition of services. Automatically finds paths between system components and creates connections.
  - Service Discovery Service (SDS): Provides a directory service mechanism (like Jini lookup service) and a mechanism by which users and programs can locate announcements across the wide area. Encompasses authentication and access control services.
  - End Clients: Fixed/mobile devices with Internet access.

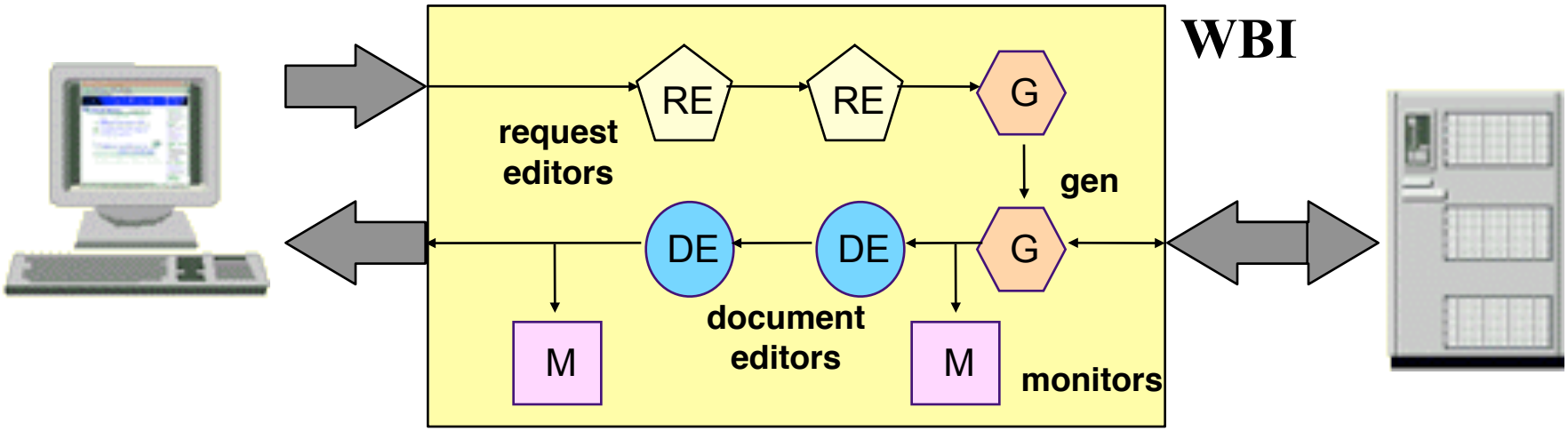
Ninja Path (ctd)

- User provides the APC facility a specification of the endpoints of the required path. The path construction process consists of four steps:
- Step 1: Logical Path Creation: The APC facility searches through operators' XML descriptions for sequences that could compute what the user requested. List of possible operator sequences.
- Step 2: Physical Path Creation: Mapping of a particular logical path onto physical nodes which execute the operators. This is done by the APC facility, which finds the lowest cost nodes that meet the requirements.
- Step 3: Path Instantiation and Execution: The APC facility starts required dynamic operators, and sets up appropriate connectors between operators. Operator nodes report problems to repair the path when necessary.
- Step 4: Path Tear-Down: When a path is no longer needed, the user informs the APC facility that it should be removed. The APC facility then stops the data, removes connectors, and shuts down any dynamic operators.

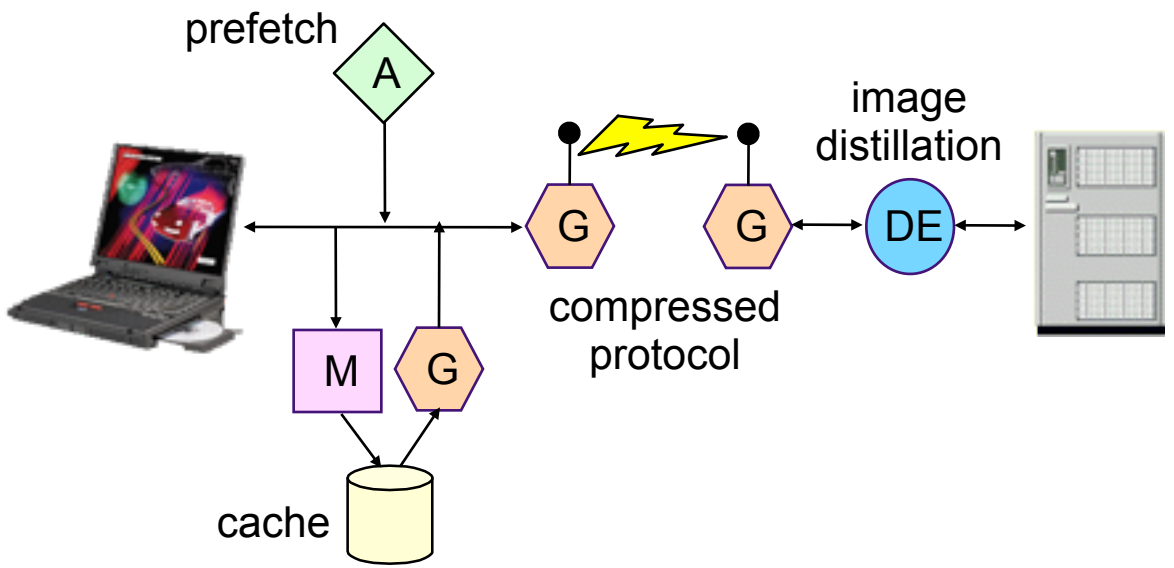
WBI (Web Browser Intelligence, IBM Almaden):

- Assembles complex intermediary systems from simpler components.
- Compositional framework:
  - Based on five building blocks: request editors [upper tier], generators [lower tier], document editors [middle tier], monitors and autonomous functions [logging and book-keeping functions].
  - These are composed into information streams, packaged as WBI plugins, and connecting origin servers with terminal devices.
  - Multiple WBI plugins can be composed into WBI services.
  - Composition of WBI plugins can be dynamic, using a rule-based approach.

WBI Dataflow



WBI Example: Wireless Web



- three plugins work together
  - cache with prefetch
  - compressed protocol extension
  - image distillation

WBI Rules

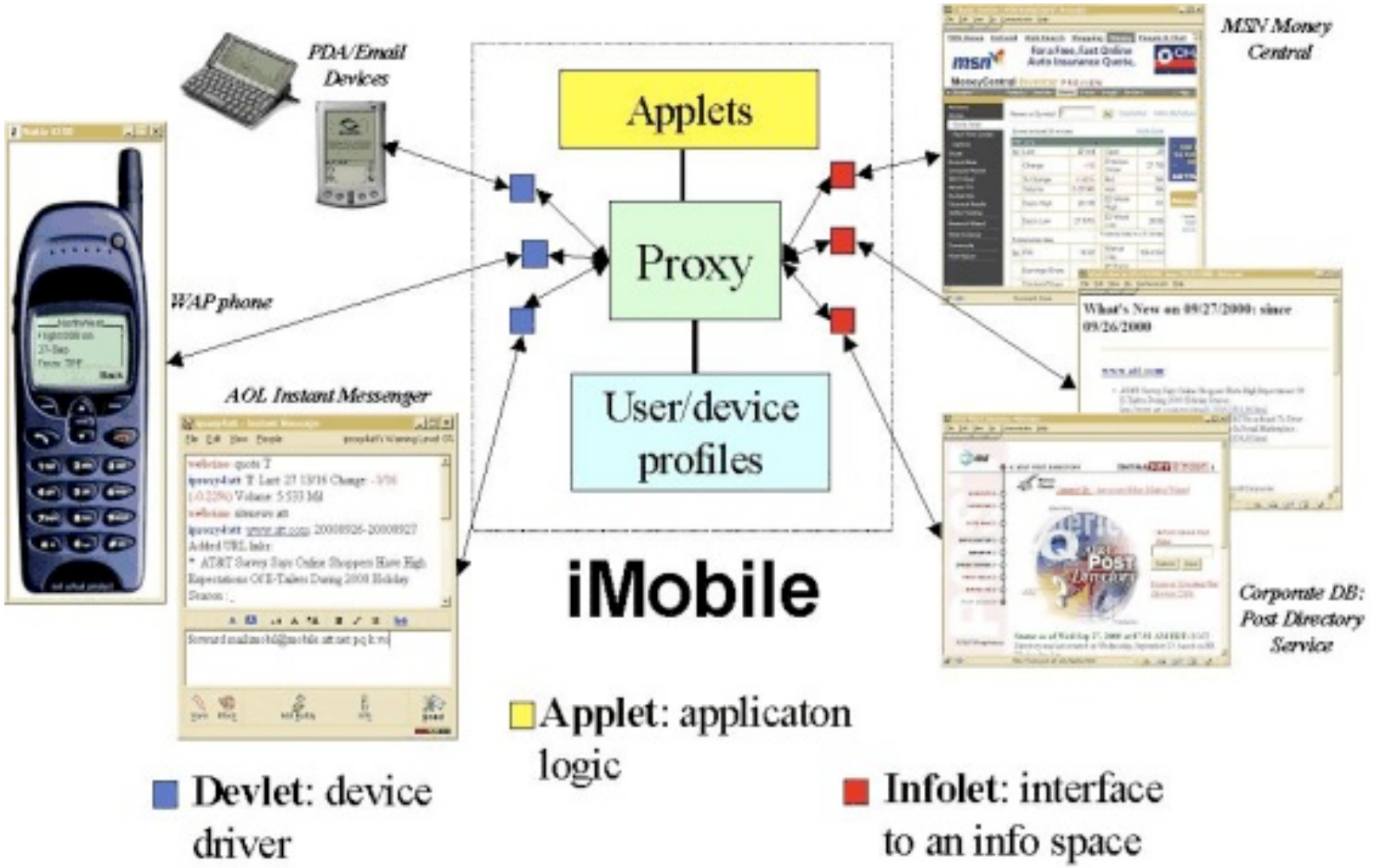
- Boolean expressions with wildcards
  - rejection allows other selection criteria
- switch on structured data in req/doc
  - URL, content-type, client addr, protocol, etc.
- priority
  - chaining (editors)
  - ordering (generators)

```
host=*.stanford.edu & content-type=text/*
```

iMobile (ATT research)

- Proxy-based platform that addresses issues in building mobile services.
- Message gateway:
  - mobile devices using various protocols on different access networks to relay messages to each other.
  - clients access internet services, corporate databases, and control various networked devices.
- Programming abstractions: devlets, infolets, applets
- The let engine, core of iMobile, implements framework for maintaining applets, devlets and infolets, supports user and device profiles for personalization and transcoding, and invokes applets and infolets to answer requests from a devlet.
- Allows new access devices and protocols to be added to its framework without changes in the service logic.
- iProxy: programmable proxy server designed to host agents and services in Java.

iMobile architecture



iMobile Devlet

- A devlet is a driver attached to the proxy that receives and sends messages through a particular protocol for mobile devices.
- Each devlet interacts with the let engine in a standard way:
  - Receive each request as a character stream.
  - Send the stream interpreted as an iMobile command and associated parameters to the let engine.
  - Return results in a MIME type acceptable by the receiving device determined by the device profile stored at the let engine.

iMobile Infolet

- An infolet hosted on iMobile uses an appropriate access method to provide an abstract view of an information space. spaces. Retrieved information may be passed to an applet for further processing.
- Examples of Information Spaces:
  - Access websites (Stock quote, weather, flight schedule, ...)
  - Corporate Database (Accessed through the JDBC and ODBC interfaces)
    - iMobile hosts a JDBC infolet that allows mobile users to access or update enterprise database information (employee data, sales data, etc) through SQL-like queries.
  - Network/Infrastructure Resources (Accessed through the CORBA interfaces)
    - iMobile hosts a CORBA infolet that allows mobile users to request services from CORBA objects.

iMobile Applet

- An applet implements service or application logic by processing information from various infolets and relaying results to various destination devices.
- It may simply sends a message from one device to another without using any information sources OR have complex interaction with other infolets.

iMobile Let engine

- Core of iMobile
- Implements framework for maintaining applets, devlets and infolets, supports user and device profiles for personalization and transcoding, and invokes applets and infolets to answer service requests from a devlet.
- All devlets, infolets and applets must be registered at the let engine before communication with other agents can occur.

iMobile Device Profiles

- Every device must register its profile with let engine.
- DEVICE NAME: protocol:acct_id
- A device profile is a list of attribute-value pairs e.g.:

dev.format.accept=text/html,/** Default profile for an email device
dev.page.size=-1

- Each device has to be mapped to a registered iMobile user.
  - limiting access to legitimate iMobile users only and
  - to personalize a service based on the user profile.
- Typical device-to-user map stored under iMobile (*rarp.ini*):

  sms:+886936731826=herman
  sms:+19087376842=chen
  mail:dchang@research.att.com=difa
  aim:webciao=chen

Infrastructures for Dynamic Content

- Intermediary infrastructures presented earlier, do not support the retrieval, execution, and caching of service code from dynamic-content providers to the intermediary infrastructure.
- In that context, several issues need to be addressed:
  - Specification and composition of intermediary services out of code components retrieved dynamically from service providers.
  - Consistency maintenance, security management, and resource management.
- Approaches in that direction focus on a three-tier architecture of a service, comprising: presentation, business logic and database tiers:
  - Active Cache system [Univ. Wisconsin].
  - Websphere Edge Services (IBM)
  - EdgeSuite CDN (Akamai) - based on the ESI specification of W3C.
  - IETF's Open Pluggable Edge Services (OPES) working group.

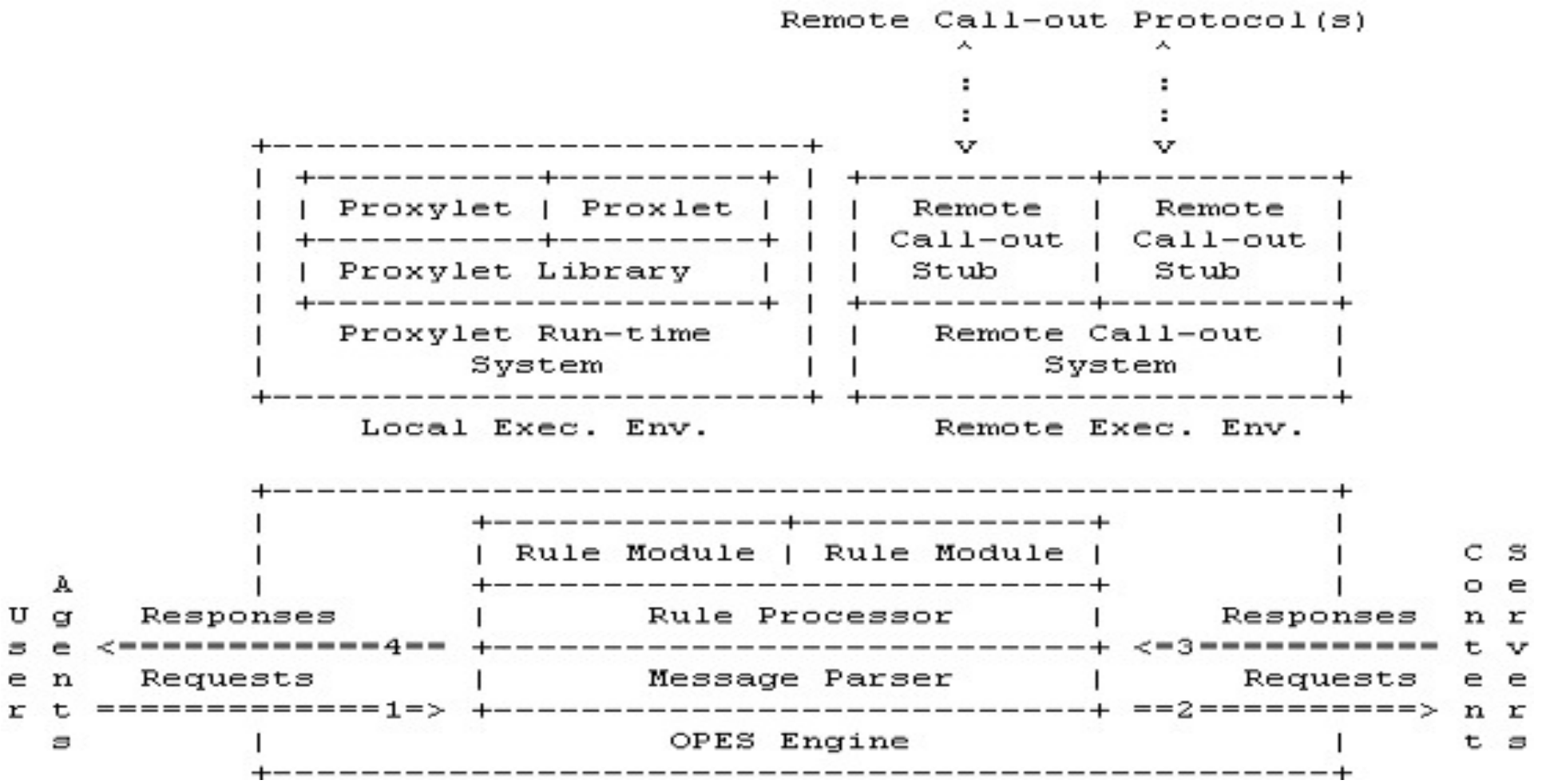
General Framework

- IETF Open Pluggable Edge Services (OPES)
  - Flows: sequence of message exchanges between client and server.
  - Entities: Processes operating on flows
    - Service applications/Proxylets
    - Data dispatcher/Engines
  - Rules: Standardized schema to encode conditions and related actions

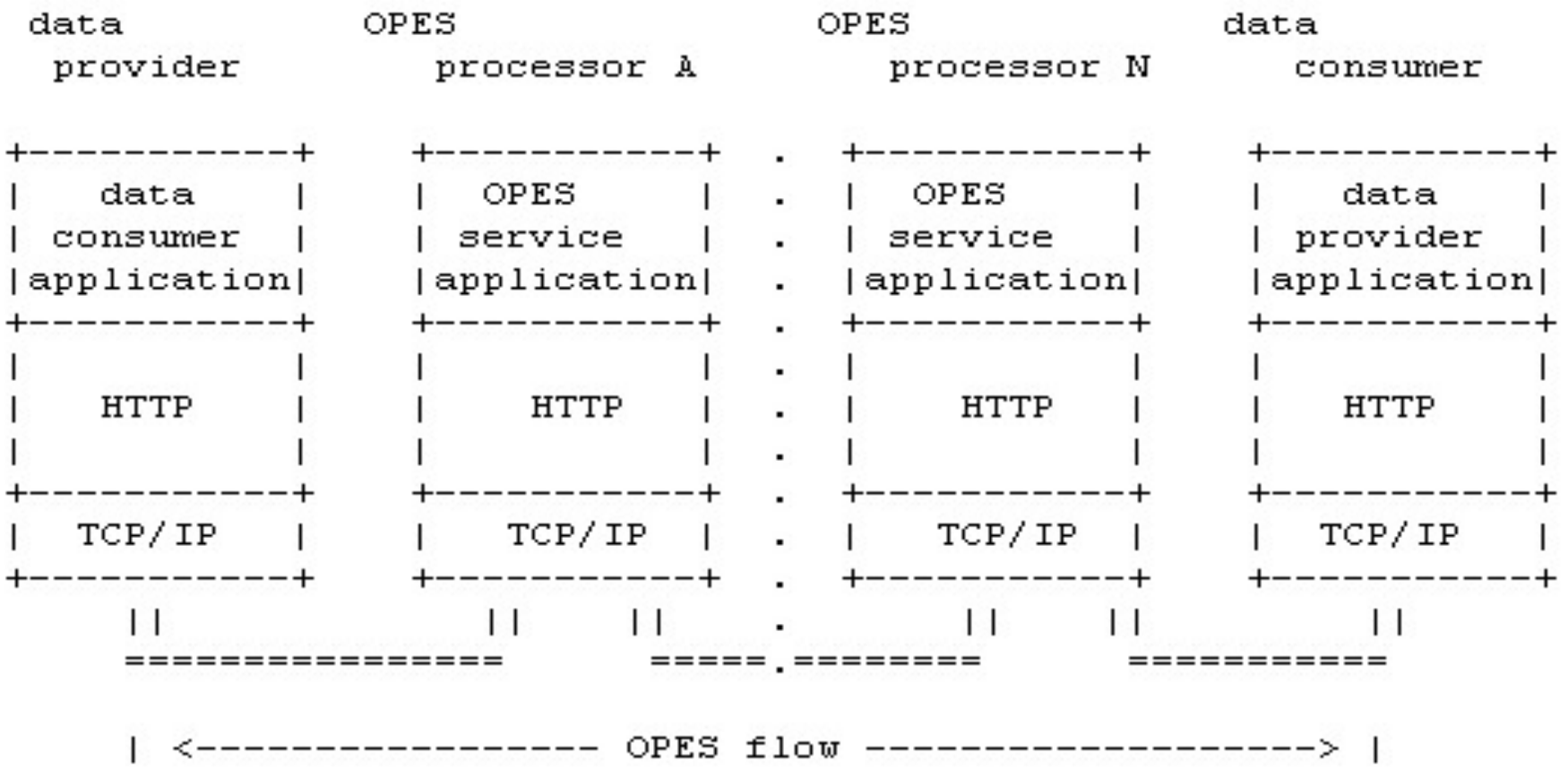
OPES architecture

- OPES entities: processes running on OPES processors deployed on the network.
- OPES flows: flows of data between OPES entities.
- OPES rules: specify when and how OPES services are injvoked

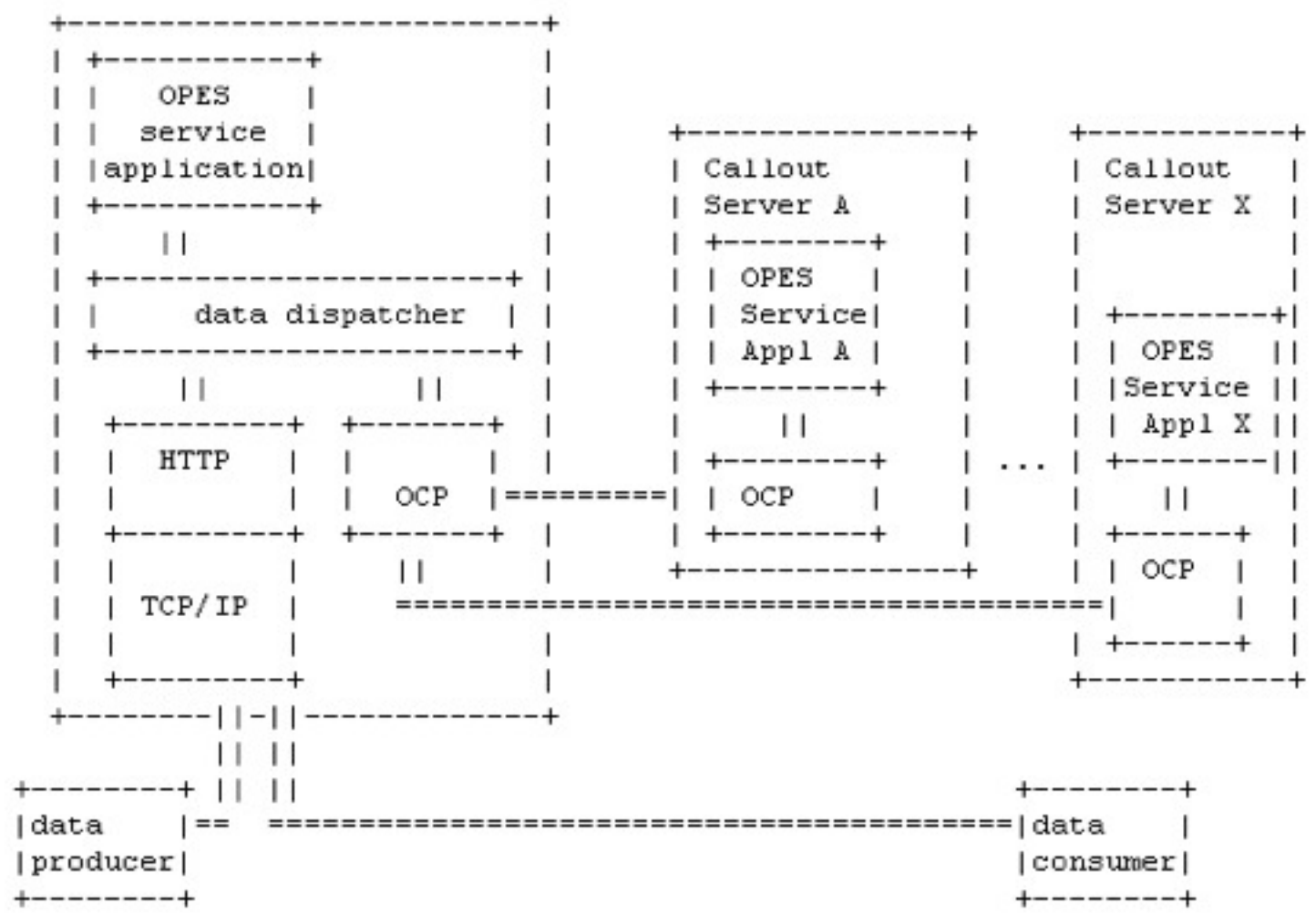
OPES Model


Remote Call-out Protocol(s)
Proxylet
Proxlet
Proxylet Library
Proxylet Run-time System
Local Exec. Env.
Remote Call-out Stub
Remote Call-out Stub
Remote Call-out System
Remote Exec. Env.
Rule Module
Rule Module
Rule Processor
Message Parser
OPES Engine
User Agents
Responses
Requests
Responses
Requests
Content Servers

OPES flows


data provider
OPES processor A
OPES processor N
data consumer
data consumer application
OPES service application
OPES service application
data provider application
HTTP
HTTP
HTTP
HTTP
TCP/IP
TCP/IP
TCP/IP
TCP/IP
OPES flow

Callout Servers



Conclusions

- Existing intermediaries lack the semantics and the system support to be used as open and dynamic overlay networks.
- Recent intermediary infrastructures offer some support for composability of services and for hosting code retrieved from remote providers.
- Many challenges still exist:
  - Programmability support: need for programming models, languages, APIs, generic compositional frameworks, execution environments, well-defined interfaces.
  - Extended Communication support: mechanisms for specifying and standardizing richer interaction semantics between intermediary components.
  - Adaptability: system support for dynamic adaptation, in terms of performance and context monitors, execution environments for execution-state migration, etc.
  - Support from Middleware: explicit description and management of adaptation conditions, resource availability; standardization of services for data management, for policy implementation.