Technical University of Vienna
Information Systems Institute
Distributed Systems Group

A Reference Architecture for Push Systems

Manfred Hauswirth
M.Hauswirth@infosys.tuwien.ac.at

TUV-1841-98-05 March 16, 1998

Popular Internet information systems like the World-wide Web still require the user to actively locate and retrieve information which is a time-consuming and tedious task. Push systems try to remedy this by reversing the communication pattern: information is actively disseminated to users. A large number of push systems is already available and these systems are gaining wide-spread use. The purpose of this paper is to present a reference architecture for such systems. It describes the general concepts, abstractions (broadcasting, channel), and components (broadcaster, receiver, transport system) of push systems. The properties and relations of components are presented and the main issues to be addressed by push architectures are discussed: scalability, network traffic, security, and e-commerce. Since push systems can allow executable content, the relations with mobile code systems are considered. A brief discussion of existing systems in respect to the reference architecture rounds out the paper.

Keywords: push systems, software architecture

Argentinierstr. 8/184-1
A-1040 Vienna, Austria
phone: +43 1 58801-4470
fax: +43 1 5058453
URL: http://www.infosys.tuwien.ac.at/

A Reference Architecture for Push Systems*

Manfred Hauswirth
Distributed Systems Group
Technical University of Vienna
Argentinierstraße 8/184-1
A-1040 Wien, Austria
M.Hauswirth@infosys.tuwien.ac.at
http://www.infosys.tuwien.ac.at/

Abstract

Popular Internet information systems like the World-wide Web still require the user to actively locate and retrieve information which is a time-consuming and tedious task. Push systems try to remedy this by reversing the communication pattern: information is actively disseminated to users. A large number of push systems is already available and these systems are gaining wide-spread use. The purpose of this paper is to present a reference architecture for such systems. It describes the general concepts, abstractions (broadcasting, channel), and components (broadcaster, receiver, transport system) of push systems. The properties and relations of components are presented and the main issues to be addressed by push architectures are discussed: scalability, network traffic, security, and e-commerce. Since push systems can allow executable content, the relations with mobile code systems are considered. A brief discussion of existing systems in respect to the reference architecture rounds out the paper.

1 Introduction

Push systems try to break up and enhance the standard pull-based interaction pattern of the World-wide Web and ease the process of information discovery and dissemination. Put simple, push systems aim at providing a service similar to TV, radio, or printed press, but enhanced with the technical capabilities of the Internet. The basic idea is the same: users inquire a "channel guide" for available information channels, subscribe to some of them according to their interests, and then continuously get information, i.e. the process of information acquisition changes from user-initiated pull to provider-side push. Instead of forcing the user to repeatedly ask for information or check whether new information has become available the user subscribes once and keeps receiving (see Figure 1).

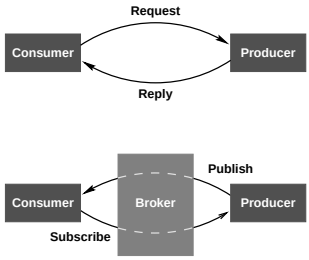

Figure 1: Pull vs. push [12]

The need for push systems stems from several reasons. The most important one is that WWW is based on a simple request/reply scheme [5], [9] that requires the user to issue a request whenever he/she needs information. This imposes a "synchronous" interaction scheme, whereas push systems would allow asynchronous information distribution: ideally, whenever information of the user's choice becomes available it gets distributed. With this admittedly idealistic assumptions, several key problems of the WWW can be addressed that push systems promise to remedy[1]:

Locating. Currently locating information is a major problem for users. Even though good search engines exist, the quality of information found is still proportional to the user's knowledge and skills (keyword selection, intuition). Push technology promises to remedy this by the concepts of information channels and subscription and by shifting the active role to the information provider.

Focusing. With push systems the user is required to state his/her preferences explicitly. Thus it is easy to provide focused information.

Customization. The user can state requirements on the data and its properties that are to be applied before it is delivered, e.g. data format, priority, keywords, etc.

Freshness. Data can actively be disseminated as soon as it becomes available. Stale data may be invalidated by the information provider.

Tailoring. Not only the user can customize the data and its properties but also the provider. The provider can control when the user sees which information. This decision

*This work was supported in part by the ARES ESPRIT Project 20477.

[1] This list is not comprehensive and only gives the key promises of push systems. A discussion of them is beyond the scope of this paper.

can be based on user profiles provided by subscription information, interest analysis, etc., and allows to tailer information to the user's profile. It is also a powerful tool for advertising.

Traffic reduction. Push systems facilitate to reduce network traffic. Trying to locate information may cause heavy traffic. Since push systems can tailor data to users' profiles, this may be decreased considerably. Additionally, an appropriate transport infrastructure can further cut down on network bandwidth, e.g. by the use of repeaters.

A broad spectrum of products with rather different properties is subsumed under the notion of "push." They differ considerably in respect to structure, components, flexibility, and interaction patterns. Their main commonality is the abstract model of automating delivery of information. The range of available push systems is difficult to evaluate and compare due to the lack of a reference architecture. This paper presents a reference architecture for push systems to provide a better understanding of the notion of push systems and to support evaluation and classification of existing products.

The paper is structured as follows. Section 2 positions push systems in the context of WWW information systems. Section 3 gives the general architecture of push systems and identifies the main concepts and abstractions. Section 4 then goes into detail and presents the architectural building blocks and their properties. Section 5 takes a closer look at the most relevant issues to be addressed by push systems. These issues critically determine the usability of a concrete system. Section 6 then relates push systems with connected concepts, e.g. mobile code systems, and tries to identify similarities and differences. Section 7 surveys current systems and gives their position in the architectural design space. Section 8 rounds out the paper by summarizing the major points made and giving a conclusion.

2 Pedigree of Push Systems

The issues addressed by push systems are not new. Several attempts were made to enhance WWW and improve the situation. A first endeavor was done with the introduction of dynamic documents [23] (client pull, server push) that pioneered the basic interaction patterns of current push systems. Search engines tried to attack the locating problem but still require advanced user skills due to the sheer amount of data they index. Customized sites, e.g. MyYahoo (http://www.myyahoo.com/), CNET (http://www.cnet.com/), etc. address the customization and tailoring issues but fall short in terms of freshness and notification.

Several of these approaches were very successful. Nevertheless, push systems are evolving rapidly since they try to comprehensively integrate all issues of efficient, focused information distribution. Many of these issues are already known from another successful information system: Usenet news [16]. The shortcomings of Usenet news compared to push systems are: inefficient *n* copy semantics, resulting in high resource and bandwidth consumption; no direct 1:1 customer-provider relation; and a difficult information structure [13]. Push systems could be considered as the modernized successor to Usenet news.

Numerous application domains have already be devised for push systems [12]. An issue of particular interest is financing of push services. Funding could either be done via requiring subscription fees, pay-channels (see Section 5), or by advertising. As mentioned above information can be tailored to users and the provider is in control of when to display information, which allows for new advertising schemes. Funding by advertisements was problematic with WWW sites since charges for advertisements have to be justified with access rates. In the presence of the current, indispensable WWW caching infrastructure, these figures present a distorted picture and are not representative. In contrast, push systems offer comprehensible access rates due to the subscription mechanism and a high degree of server-side control.

3 General Architecture

The previous sections gave a quick overview on push systems, the basic ideas, and their evolution and applicability to set the stage for an architectural analysis. Software architecture [25], [10] is a formal arrangement of architectural elements and describes key design idioms. It provides the framework within which to satisfy the system constraints and provides both the technical and managerial basis for the design and implementation of systems [25]. It supports understanding of important system concepts, system structure, interrelationships and communication, and helps identifying components for reuse.

No standard definition of software architecture exists. A frequently used definition is provided in [10]:

> *[Software architecture is a level of design that] goes beyond the algorithms and data structures of the computation: designing and specifying the overall system structure emerges as a new kind of problem. Structural issues include gross organization and global control structure; protocols for communication, synchronization, and data access; assignment of functionality to design elements; physical distribution; composition of design elements; scaling and performance; and selection among design alternatives.*

The elements of an architectural description are components, connectors, and (architectural) configurations (see e.g. [21]). The following definitions of these elements are based on [21]. Components may be units of computation or data stores. They can be single procedures up to entire applications. Connectors model interactions among components and rules that govern the interactions. They may

be message routing devices, shared variables, or protocols, etc. Configurations are connected graphs of components and connectors that describe architectural structure. Together with models of components and connectors, descriptions of configurations enable the assessment of concurrent and distributed aspects of an architecture.
Figure 2 shows the general architecture of a push system.

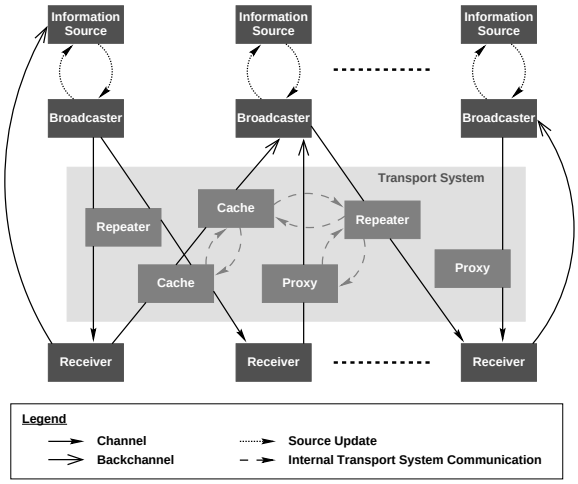

Figure 2: General architecture of push systems

As a first step let us consider a simple push system and abstract from the transport system. In this case a push system consists of information sources, broadcasters, and receivers. These architectural components interact via the channel and source update connectors. The information source component models the data inputs of the push system which are to be disseminated. It can be any kind of data source. Via a data exchange interface modeled by the source update connector it feeds data into the broadcaster component and receives feedback (notifications) from the broadcaster. The broadcaster is in charge of controlling and scheduling the dissemination process. It distributes data via a set of channels to receivers. A receiver can receive multiple channels from multiple broadcasters.
The channel and source update connectors define the protocols between the components. The basic interaction pattern in this setting is downstream distribution from the information sources to the broadcasters via channels to receivers with little to no upstream communication. Upstream data flow only appears if it is necessary for satisfaction of the connectors' (protocols') requirements, e.g., flow control.
This simple model can be enhanced by the introduction of bidirectional channels, i.e. upstream data flow from the receiver (user) to the broadcaster or to the information source. Instead of bidirectional channels the conventions channel, for downstream, and backchannel, for upstream communication, shall be used in the following. Although channel and backchannel can physically be integrated in a single bidirectional channel in the implementation, conceptionally they may be completely unrelated and thus are modeled as two separate connectors. The purpose of the backchannel is to communicate receiver/user information to the data sources and broadcasters and allows those to do adaptations of the data quality, gather data for the adaptation process, respond to user requests, add functionality, etc.
Such a simple system may already be suitable for small to medium size intranets. For bigger intranets, segmented intranets that need special functionality, or the Internet, however, this architecture is inadequate. These application domains require the introduction of a dedicated transport system level. Though the transport system introduces levels of indirection into the architecture it is necessary to achieve performance, scalability, and flexibility in a push system. The transport system shall be transparent towards channels and backchannels as indicated in Figure 2.
The transport system consists of caches, repeaters, and proxies. These components cooperate via specialized transport system connectors, i.e. protocols, that are internal to the transport system and are not visible outside. Caches and repeaters shall help to conserve network bandwidth by reducing the load on broadcasters and bringing channel data "closer" to the receivers. In case of a cache this is done on-demand, whereas a repeater is preloaded. Proxies model situations where no direct connection to a push system component is possible, e.g. for security reasons, to control network traffic, etc. The proxy component thus acts on-behalf of some other components.
Putting the key components together then defines the architecture for push systems as depicted in Figure 2. The next section will give a more detailed view of this general architecture. In the following the emphasis of the architectural analysis will be on components and connectors. Though Figure 2 may be viewed as a high-level configuration, specific configurations are not part of a reference architecture and thus beyond the scope of this paper.

4 Architectural Building Blocks

The key architectural building blocks of a push system are channel, broadcaster, receiver, and transport system. A model for the information source can easily deducted from the other components and thus is not described explicitly. Internal transport system connectors shall not effect the architecture assuming that the transport system itself is transparent for the other components. Many approaches for the necessary transport-level protocols (caching, coherence, data exchange, etc.) exist in the literature. Thus they are not described in detail here.

4.1 Channel

A channel is a connector between a broadcaster and a receiver. It determines the protocols between these components, e.g. channel access protocol, subscription protocol, etc. A channel can exist in multiple instances connecting multiple broadcasters with receivers. It determines sev-

eral properties of the data and the supported functionalities, i.e. the quality of service (QoS):

Type of information. Like with TV and radio, content that can be obtained from a channel is usually focused to a specific topic, e.g. weather reports, product support, news, etc.

Data format. The type of information and the protocols to access the channel define the content types that are possible for the channel. This can be static data, e.g. text files, pictures, data, etc., dynamic content, i.e. executable programs, streaming data, e.g. real-time audio or video, or combinations of these types.

Personalizing. This channel property determines the extent of user customization possible for the channel (content selection, operation modes, interaction, etc.)

Content expiration. Content can be transient or persistent. An expiration strategy for the channel must exist in order not to fill up the user's disk space.

Update strategy. This is closely connected to content expiration but primarily defines the way content updates are done for the channel. In the simplest case the strategy is to replace information without reusing it. A more efficient way are differential updates where possible (this heavily depends on the type of information, data formats, etc.). An important issue is the update frequency which has major impact on data accuracy, network traffic, and scalability.

Scheduling strategy. Channels can either be time-scheduled or content-scheduled. Time-scheduled channels (TSC) deliver "unrepeatable", "life" content. Depending on the time at which the user accesses the channel the current "broadcast" is delivered (comparable to TV, radio). To make content accessible at a later time a virtual channel recorder and player (VCR, VCP) are necessary. Content-scheduled channels can deliver content "independent" of the time line, e.g. a news services that also offers back information.

Operation mode. For the users it is important when and how information is delivered. Online operation is not always necessary or possible. A user may want to load a mobile computer with a channel and operate it offline. For this mobile computing support feasible synchronization protocols must exist.

Payment. Currently most channels are free. It is foreseeable, however, that several future channels will involve payment (support channels, special contents, etc.) Several payment schemes are possible: pay-per-view, content-based, time-based, flat fee, etc.

As mentioned above not only downstream information flow but also a backchannel may be necessary for some channel types. A backchannel models data flow and interaction from the receiver to the broadcaster, the information source or a third party. It may use the same protocols like the corresponding channel. Since push systems are usually oriented more towards downstream communication, frequently a lighter version of the communication facilities or a different medium are used, e.g. a streaming data channel may have an HTTP-based backchannel. Nevertheless, the closer and more seamless channel and backchannel are integrated the better. The conceptual user interface will become simpler to use and more transparent to the user if tight integration exists. Backchannels can exist on a per-channel basis, for a set of related channels, or for the full set of channels available from one broadcaster. Another important difference between channels and backchannels is the type of relationship between the parties. Conceptually a channel is a 1:n relationship, whereas a backchannel is 1:1. Some of the issues described above can be modeled as depicted in Figure 3: the channel defines a data stream that is directed through a set of configurable filter components. Filtering can both be done by the receiver and by the server. These components fulfill some of the functionalities described above or add others (value added services), e.g. payment, content selection for children, etc.

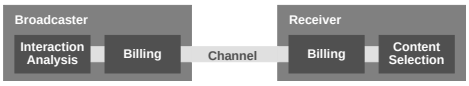

Figure 3: Channel structure

A comprehensive list of features which a channel can support is given in [29].

4.2 Broadcaster

A push system has at least one component that offers channels and distributes channel contents to the channels' subscribers. In analogy to TV this component is called broadcaster. For small-scale intranet applications, like company information systems, one dedicated broadcaster may suffice. For large-scale applications that provide channels to a possibly huge number of subscribers (thousands of receivers) it cannot be a single component. To scale a push system to such a size a broadcasting infrastructure as we know it from TV must exist.

The broadcaster itself may be distributed. A set of broadcasters may provide the channels and exchange updates among each others to stay in sync. Standard techniques like primary copy, process groups, or multicasting for distribution, replication, and consistency can be applied here. Example architectures are:

Primary broadcaster. This architecture consists of a primary broadcaster plus a set of hierarchically arranged first, second, etc. level broadcasters that bring channels closer (in terms of bandwidth, delay, geography, etc.) to receivers. The functionality of broadcasters aside from the primary broadcaster is basically the functionality of a repeater: the content that is received is sent out again with no further contents or functionality added. This is the architecture of TV broadcasting (the difference in the notion of broadcasting is neglected here).

Partitioned broadcasters. The broadcasting system consists of a set of active broadcasters that for each chan-

nel provide part of the contents and functionality. Nevertheless, to the clients this system provides the illusion of a monolithic entity. Internally a synchronization strategy must be used. Such architectures can vary considerably in the type and degree of distribution. This architecture is known from other large scale systems like X.500 [7] and adds another magnitude of complexity to the system.

Simple Broadcasting. The broadcasting system does not take care of the transport and distribution of channel contents but relies on functionalities of the transport system. The simplest pattern is a single broadcaster that relies on a caching infrastructure inside the transport system. Most of currently available products follow this approach. This approach may seem unrealistic but nevertheless is already applied successfully by other systems like the WWW.

Real systems may of course be combinations of the approaches above. The primary goal in all the above settings is that receivers can access channels from a broadcasting component that is "close" to them in some respect (bandwidth, delay, geography, etc.) to minimize network traffic, provide short delays, and allow scalable systems.

4.2.1 The Notion of Broadcasting

So far the notion of broadcasting in the context of push systems has been used shallowly. Broadcasting in push systems cannot rely on a medium that offers broadcasting functionality like for TV or LANs (e.g. Ethernet). The medium for a push system is the Internet. Unlike LANs, e.g. Ethernet, there is no broadcast address concept that could be exploited. On the one hand this is due to the extremely heterogeneous structure of the Internet. On the other hand such an Internet broadcast facility is not even desirable. Imagine the amount of additional traffic such a concept would produce on the Internet. Actually the desired functionality is closer to a groupcast/multicast functionality since only subscribers should get the information. On the Internet, however, this does not make a big difference. The impacts on network traffic would still be too big. Additionally, new and complex groupcast/multicast protocols for large-scale use which insure timely distribution to all subscribers would be necessary.

Thus push systems usually use a higher-level abstraction of broadcasting: a broadcast—or more correctly a multicast since only subscribers receive it—is emulated by lower software layers. Several strategies for this kind of emulation are possible:

Client poll. At regular, user-definable intervals the receiver checks with the broadcaster whether the receiver's view of the channel is still consistent or needs to be updated. This pattern actually puts the above concept of broadcasting upside-down: the initiative changes from server-side to client-side. The main disadvantages of such a scheme are that complete data accuracy cannot be achieved (or only at the cost of very high polling frequencies which induces high network traffic) and a possibly high number of unnecessary messages if the channel is not very dynamic (or only dynamic on client initiative). But on the other hand consistency requirements of channels are usually rather relaxed and polling frequencies between 10 minutes and a day do not cause high network traffic. Additionally, polling messages are likely to be rather small (some 100 bytes). The remaining drawback, however, is notification (freshness of data): how can the receiver be notified of high-priority changes that happen during its polling interval or situations which require immediate attention. Polling, however, is frequently used in push systems since it is robust, simple to implement, allows for off-line operation, and scales well to high numbers of subscribers.

Server push. In case of updates the broadcaster contacts its subscribed receivers and sends new content. For large-scale push systems this approach has several drawbacks. Since no broadcast infrastructure similar to TV or LAN broadcasts exists, receivers must be contacted in a different way. Contacting the receivers sequentially does not even scale for medium numbers of subscribers. It would be too time-consuming, and thus leave receivers with different views of channel information depending on their ordinal number in the pushing process. Using multicasting would work up to a certain number of users but would not scale to high user numbers. So most likely a hierarchic infrastructure would have to be used, e.g. organizations announce a dedicated host which the broadcaster can contact and this host then takes care of further dissemination. Server push broadcasting also requires a directory of subscribers to be contacted. Such a directory imposes additional effort since it must be maintained and kept consistent. For client poll broadcasting such a directory is only optional. Generally scalability of server push will be lower than client pull's but may allow better consistency and freshness of the channel content. Another problem of server push broadcasting is that receivers may not be online all the time, i.e. offline operation. Loss of data is possible which must be compensated by re-broadcasts, if possible. This coherence issue considerably adds to the server's load and complexity.

A promising approach is to combine the two strategies into a hybrid scheme: server push is used to distribute information on the data available in channels (references) allowing the receivers to keep track of available content, thus having good freshness, notification, and consistency properties; client poll then is used to request the actual content if wanted.

An issue interwoven with the broadcasting algorithm is the channel update strategy. The question is how changes in the data are transmitted to the receiver. A plain replacement strategy would cause high network traffic in the face of possibly little changes in the data. A better way to deal with updates is to use a differential/incremental strategy. This conserves network and computing resources and remedies the problems of network partitions and offline operation.

4.2.2 The Notion of Subscription

As described above receivers cannot simply "stick their antenna into the air" and receive channels. To receive a channel the client usually has to run through a subscription process beforehand, though this is not necessary per-se, e.g., in the case of free channel access and client poll. Nevertheless, a subscription is frequently required.
The subscription process usually requires the user to lookup a channel directory (channels + descriptions) to select channels, give some personal information, and provide his/her profile of interests. The channel directory can be made available as a default channel for the new subscriber to keep him/her informed of changes. The broadcaster on the other hand may need subscription information to learn about the receiver's destination address, network connection, etc.
Information about the subscriber besides the distribution address is not necessary for technical reasons. Channel suppliers and broadcasters, however, frequently require subscription information for economic purposes: they want to establish relations to their customers and need to fund their channels. Funding will be done most likely via advertisement. Since pricing of advertisements heavily depends on the range and readership of a medium this information is vital for the supplier. Advertisement may help to keep channels free of charges and attract users. Regardless of one's opinion towards advertisements technical precautions for supporting them have to be taken. May this option be used or not, it will definitely be important for a successful push product.
Of course privacy and security of subscribers must be guaranteed in such a setting. Ideally users should be able to specify what of their private information can be handed to other companies and that the storage where such information is kept is secure, e.g. by encrypting this data.

4.3 Receiver

The broadcaster's counterpart on the client side is the receiver. If we abstract from the transport medium only the broadcaster and receiver interact. The receiver has two main components: channel access and user interface. The receiver is the interface that facilitates interaction between users and channels. It gets channel data from broadcasters and presents it to the user. It allows the user to manipulate, control, and customize the user profile, the received information, and the channels. According to a channel's defaults and the user's settings the receiver is responsible for updating (received/requested) channel content, expiring channel data, and freeing disk space on demand.
Finding channels can be implemented in several ways. The receiver could query a channel directory or present information on available channels as a (meta-)channel itself, which would perfectly fit into the architecture. Upon subscription of a channel the receiver has to know the channel's broadcaster. Ideally the receiver should be able to find out about an optimal broadcaster for a channel, i.e. use the "fastest", "closest" broadcaster. To enable this choice, the subscription process must make appropriate data available to the client. Consequently a receiver will communicate with several broadcasters. The properties and requirements for these interactions can directly be derived from the previous sections. If a backchannel is available the receiver is also in charge for handling this communication path and providing the user with a user interface.
The channel metaphor can also be applied to the receiver itself. Once the receiver software is installed it can also become a channel, i.e. updates of the receiver software are downloaded automatically as soon as new versions become available and installed if the user agrees. Thus users would only have to do an initial setup and could use new software versions immediately. Bringing this concept into operation would have tremendous impacts on software distribution and maintenance. While currently backward compatibility and distribution of software are a major concern when upgrading a widely-used software this costs could be reduced considerably with this concept.
Channels can also hold dynamic content, i.e. executable code. In analogy to WWW/Java (applet) this content type shall be termed *pushlet*. A pushlet is executable code and data which is intended for execution at the client. Pushlets execute inside a user-configurable environment provided by the receiver. This environment supplies the necessary runtime system. The languages for pushlet code and the environments implied by that may differ considerably. The following requirements, however, must be addressed regardless of the concrete system:
Code authentication. The environment must ensure that the dynamic content is authentic (code signatures).
System protection. The environment must protect the client system from malicious code that endangers the client's system integrity.
Authorization. The environment must offer a flexible user-definable authorization scheme, i.e. the user must be able to allow/disallow certain operations for pushlet code. E.g., some pushlets may require disk access, network access, etc. to fulfill their functionality. For the sake of flexibility it should be up to the user to decide whether to allow such operations.

4.4 Transport System

So far we have completely abstracted from the transport system when discussing the broadcasting process, broadcaster/receiver interaction, etc. In a large-scale setting, however, a dedicated transport system is necessary to make a push system scale and operational, i.e. take care of efficient network access in terms of decreasing network bandwidth consumption and increase availability and responsiveness. This section identifies its main components and connectors.

A key design issue for the transport system is transparency towards the components and connectors described in the previous sections. Transparency, however, can only be achieved to a certain extent. Both the broadcaster and the receiver need certain knowledge on the transport system to be able to use it. E.g., they must know about the concept of repeaters, so that the broadcaster can feed them and the client can connect to a "close" one. This influence, however, should be minimized.

The main connectors in a push system can be derived from the previous sections: subscription protocol, channel protocol (differential updates), backchannel protocol. The transport system should be as transparent as possible towards these connectors.

The components of the transport system can be modeled by a so-called *base distribution component* (BDC). A BDC is a generic component that acts as a broadcaster towards receivers and as a receiver towards broadcasters in order to achieve the transparency property. Thus it must understand the concerned protocols. Besides these protocols other connectors among BDCs may exist.

A BDC can exist in several configurations: repeater, cache, and proxy.

Repeater. Clients shall not automatically connect to the primary provider but can get redirected a repeater. A repeater is preloaded with the channels' contents and offers the same data as the broadcaster but is "closer" (in terms of network properties) to the receiver.

Cache. Basically a cache is the same as a repeater but is not preloaded (on-demand repeater). A cache only loads content on a receiver's request.

Proxy. A proxy facilitates access to channels where receivers cannot gain direct access, e.g. receivers may be located behind a firewall. It acts on-behalf of the receiver. Every proxy has a domain translator sub-component that translates back and forth between the generic proxy functionality and the application domain functionality, e.g. in case of a firewall it translates between the firewall requirements and the push system requirements.

BDCs can be organized according to some pattern inside the transport system. Without constraining the applicability let us assume this organization is hierarchical. Then BDCs can communicate both at the same level and between different levels. This is meaningful for efficiency considerations, e.g. consider a caching infrastructure where a request is sent to all caches at a certain level to maximize the probability for a cache hit; if no hit occurs the request is forwarded up the hierarchy. A specialized connector, a so-called transport system coherence protocol (TSCP), is necessary for these purposes. TSCP is in charge of exchanging meta-information on the data among BDCs and to initiate data transfers if necessary (both preloading and caching should be possible).

5 Architectural Key Design Issues

Among the set of architectural facets of push systems some are vital and should be addressed by concrete architectures. This section discusses such key issues in a push system's architecture.

Scalability. As long as a push system is intended for a rather small set of users and operates on a homogeneous network infrastructure scalability is not of high importance. But on a heterogenous network like the Internet and with high numbers of users it is. The key issue for a push system is how to manage a high number of subscribers and satisfy their requirements in terms of freshness of information, customization, tailoring, etc.

A well-designed and efficient transport system as described in Section 4.4 is the first step towards scalability.The transport system must minimize resource consumption and network traffic, and distribute load while maintaining the properties of a good push system. A key requirement for achieving this is cooperation among the transport system components and between the different organizations running them. As an example for such an organizational structure repeaters and caches could be co-located with Internet service providers that every user needs anyway. This would set up an efficient initial transport system infrastructure. For efficiency, proxies and caches could also be co-located.

To have the transport system and the receivers cooperate efficiently, information services (traders, brokers, directories) to answer repeater requirements are needed. E.g., a receiver would need information where to find the "best" (in whatever respect) broadcaster for a certain channel. Protocols for such purposes are currently being defined, e.g. [15], and should be included into push systems.

Beside these domains related to networking, scalability is also an issue aside the transport system. Management of high numbers of users also binds reasonable resources at broadcasters, e.g. maintenance of subscriptions, user profiles, tailoring of information, etc. A push system should define processes for these management areas and support them with software tools.

Electronic commerce. Push systems like other media involve costs for the operating organization. Funding can either be done by advertisements or by collecting fees for the use of channels. A push system's architecture should offer flexible support for electronic commerce to accomplish this.

If push systems succeed a market for advertisement-free channels will develop. Additionally, professional services (stock figures, news agency information, etc.) usually are only offered for payment. A demand for flexible payment schemes will then arise: flat fees, pay-by-use, pay-by-time, pay-per-channel, pay-per-information, etc. Like with pay-TV a channel could offer a certain percentage of "clear" information that can be received free of charge to attract customers and other information in an scrambled or encrypted

way which only is readable for subscribers. The number of possible models is unbounded.

The support of electronic commerce has major impact on the architecture of a push system, e.g. security, confidentiality, identification of subscribers, payment, management of accounts, etc.

Security and Privacy. Content distributed by a push system can be executable (pushlets), i.e. code that is intended for execution at the receiver's site. The receiver must provide a security architecture to protect the client from malicious code, i.e. prohibiting intrusion, eavesdropping, or other damages. Pushlet code must be authenticated to prevent code tampering. The security model could base on existing ones, e.g. Java's [11], [20] security manager architecture, and should allow flexible configurations. The user may want to allow certain operations to trusted pushlets. For this purpose the security model should include a user-configurable, semi-automated security negotiation process.

Maintenance. As a push system evolves changes in the software may be necessary that may cause incompatibilities. This is an issue especially for receiver software, since it has the biggest installation base. To support seamless upgrades the receiver could be made into a channel itself. With well-designed architectures this could free providers from many software maintenance issues. Such an architecture must pay special attention to security, e.g. who is allowed to change the receiver software, or availability issues, e.g. what if the upgrade fails.

Openness. Currently available push systems have a big disadvantage: they are proprietary, i.e. incompatible. This means that users are tied to one system or dedicated clients have to be installed for each push system. The same holds true for providers. Neither can data be exchanged between the available products nor can transport infrastructures be reused. This may be reasonable during the startup phase of a new concept but later support for interoperation should be included. This may either mean adaptors or open protocol standards. Integration efforts are currently under development, e.g. [14].

6 Related Concepts

Some of the new paradigms that try to enhance and modernize Internet services are related to push systems. In the future they may coexist or merge with push systems:

Mobile code. Program code is no longer tied to a processor/operating system. The concept is to have code travel around networks and computers to fulfill their task. Code can be mobile in terms of portability, e.g. Java [1], or in terms of functionality, e.g. mobile agents [30]. Mobile agents are a promising approach to address information discovery, brokering, or offline operation.

A push system could be seen as a mobile code system and vice versa: a push system that distributes pushlets is a special form of a mobile code system. If the concept of push is considered as a distribution infrastructure for pushlets and every receiver in a push system is also a broadcaster to forward pushlets, then we have the basic architecture of a mobile agent system. A mobile code system on the other hand can be used to actively transport information to users (receivers) and thus can serve as a push system, e.g. [28].

Push systems, however, are data-centric, focusing on efficient dissemination of information, whereas mobile code systems are functionality-centric, dealing with the distribution of computation to reach a defined goal. Key issues for push systems are efficient distribution, management of high numbers of users, and electronic commerce. Key issues for mobile code systems are security, authentication, authorization, monitoring, and feedback.

Ubiquitous Computing. A user's personal computing environment is available wherever he/she may be, e.g. [31]. This comprises physical mobility of computing devices as well as availability of user data, user configuration, and applications, i.e. "dissemination," be it real or virtual, of these data.

The application domains of push systems, mobile code systems, and ubiquitous computing overlap to some extent. They all try to solve similar problems to a different extent and in disparate aspects, e.g. information discovery, dissemination of information, availability of resources, etc.

7 Current Systems

To relate the architectural considerations presented so far to real systems, this section discusses current systems and points out their main aspects. A detailed discussion of these systems would be beyond the scope of this paper.

Castanet [18], [19] is a Java-based push system with a client-poll broadcasting scheme. It supports static content and pushlets, offers user-definable channel plugins (monitoring, tailoring), and has an elaborate transport system infrastructure (proxies, caches, repeaters). The receiver is considered to be a channel itself and thus can be maintained automatically by the system.

Netcaster [24] is basically Castanet, which was licensed by Netscape, bundled with Netspace's web-server channels. A web-server channel is a URL that is recursively checked and downloaded by the receiver at regular intervals. It can hold Java applets that are executed at the client.

PointCast [27], [26] offers a sophisticated advertising infrastructure, an easy-to-use GUI, and tools for creation and management of channels. It supports static content and uses client-poll broadcasting. The transport system has caching managers for client sites to reduce network traffic.

BackWeb [4], [3] uses client-poll broadcasting and offers user-definable channel hooks (monitoring, tailoring). It supports static and dynamic content, i.e. Java applets that are executed at the client site using an external Java Virtual Machine. Server-push broadcasting is possible with third-party products. Channels have a differential update

strategy. Support for flexible content description and presentation (BackWeb Authoring Language Interface [2]) is provided. The transport system has proxies and caches for client sites with multiple receivers.

Webcasting [22] is Microsoft's push concept. It uses client-poll broadcasting. Channels are defined in the Channel Definition Format (CDF) [8] as collections of URLs that are checked on a regular basis for updates.

WebCanal [17] is a time-scheduled push system for the dissemination of HTML pages (the above systems are content-scheduled). A special multicast protocol, Lightweight Reliable Multicast Protocol (LRMP), allows server-push broadcasting.

Intermind [29] offers a client for polling channels provided by Web site publishers. It has no dedicated broadcaster or transport system. Its central concept is the so-called channel object that encapsulates data, metadata, methods, and rules for a channel. Suggested channel feature categories are: profile exchange, negotiated delivery, negotiated security, active message processing, channel feedback, channel linking, and data interchange.

Keryx [6] is a notification system. Notifications do not include content themselves. For content distribution a hybrid push-pull scheme is suggested: clients describe events they want to be notified upon; in case such an event occurs clients are actively notified (push); a notification can hold information where the client can retrieve the actual content (pull).

Common Datacast Architecture (CDA) [14] is not a push system per-se but an approach to bridge the gap between information providers and channel transmitters. A channel builder component creates subject-based information channels, connects those channels to data-sources and transmits that information through various channel transmitters, i.e. push systems. The long-term goal is an architecture for customized information subscription, publication, and dissemination.

8 Conclusion

Push systems are a promising new technology for information dissemination. Their basic approach is active distribution of data. Data can be static or dynamic, i.e. code to be executed at the client (pushlet). Since it is a rather new paradigm many approaches exist but a unified model, terminology, and architecture are missing.

The main contribution of this paper is the definition of a reference architecture for push systems to allow for classification and comparison of push systems. It introduces a clear terminology and describes the architectural building blocks and their relations. The main elements identified are broadcaster, receiver, and transport system. The significant concepts are channels and the notion of broadcasting.

Ideally the transport system is transparent towards the other components and the broadcasting process though in reality this assumption does not hold. A closer look revealing the transport system's internals—repeaters, caches, proxies, and their cooperation—is necessarily given since it has major impact on the concept of broadcasting. The notion of broadcasting in push systems was discussed and two groups of broadcasting algorithms were identified: client poll and server push. Key architectural issues to be addressed by well-designed systems are scalability to large numbers of users, security, and electronic commerce.

With the reference architecture presented in this paper the broad spectrum of available push systems can now be classified and evaluated. This facilitates comparison of push systems and supports users in choosing a concrete product that fits their requirements.

Acknowledgements

I would like to thank Harald Gall for his comments on this paper and our fruitful discussions.

References

[1] K. Arnold and J. Gosling. *The Java programming language*. Addison-Wesley, Reading, Mass. and London, 1996.

[2] BackWeb. *BackWeb creative guide*. BackWeb, 1997. BALI (BackWeb Authoring Language Interface) reference guide.

[3] BackWeb. *BackWeb Server Guide*. BackWeb, 1997.

[4] BackWeb. BackWeb – a cooperative architecture for a flexible push-pull broadcasting solution. Technical report. BackWeb, March 1997. http://www.backweb.com/pd/whitepaper.html.

[5] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. Technical report RFC1945. InterNIC, Network Working Group, May 1996. ftp://ds.internic.net/rfc/rfc1945.txt.

[6] S. Brandt and A. Kristensen. Web push as an Internet notification service. Technical report. Hewlett-Packard Laboratories, Bristol, UK, 1997. Keryx. http://keryxsoft.hpl.hp.com/doc/ins.html.

[7] D. W. Chadwick. *Understanding X.500 – The Directory*. Chapman & Hall, 1996.

[8] C. Ellerman. Channel Definition Format (CDF). Technical report. Microsoft Corporation, September 1997. http://www.microsoft.com/standards/cdf.htm.

[9] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. Technical report RFC2068. InterNIC, Network Working Group, January 1997.

[10] D. Garlan and M. Shaw. An introduction to software architecture. In V. Ambriola and G. Tortora, editors, *Advances in Software Engineering and Knowledge Engineering*, volume 1. World Scientific Publishing Company, 1993.

[11] J. Gosling, B. Joy, and G. Steele. *The Java language specification*. Addison-Wesley, Reading, Mass. and London, 1996.

[12] R. Hackathorn. Publish or Perish. *BYTE*, **22**(9):65–72, September 1997.

[13] M. Hauswirth and T. Gschwind. Modernizing Usenet infrastructure – a cache server for Usenet news. Technical report TUV–1841–97–15. Distributed Systems Group, Technical University of Vienna, September 1997.

[14] M. Hebert. *A push in the web direction*. The MITRE Corporation, July 1997. Common Datacast Architecture (CDA). http://www.mitre.org/pubs/edge1/july_97/fourth.htm.

[15] C. Honton. *Service Discovery Protocol*. InterNIC, December 1997. ftp://ds.internic.net/internet-drafts/draft-honton-sdp-01.txt.

[16] D. Lawrence and H. Spencer. *Managing USENET*. O'Reilly & Associates, Incorporated, January 1998.

[17] T. Liao. WebCanal: a multicast web application. *Sixth International World Wide Web Conference* (Santa Clara, California, USA, April 6-11, 1997), April 1997. http://webcanal.inria.fr/webcanal/www6.html.

[18] Marimba. *The Castanet product family*. Marimba, Incorporated, 1997. http://www.marimba.com/doc/general/2.0/introducing/introducing.html.

[19] Marimba. *Developing Castanet channels*. Marimba, Incorporated, 1997. http://www.marimba.com/doc/Castanet_Developer_Docs/2.0/index.html.

[20] G. McGraw and E. W. Felten. *Java security: hostile applets, holes, and antidotes*. John Wiley, New York, 1997.

[21] N. Medvidovic and R. N. Taylor. A framework for classifying and comparing architecture description languages. *Proceedings of the 6th European Software Engineering Conference* (Zurich, Switzerland, 22–25 September 1997), pages 60–76, M. Jazayeri and H. Schauer, editors. Springer Verlag, Berlin, 22–25 September 1997.

[22] Microsoft. Webcasting in Microsoft Internet Explorer 4.0 White Paper. Technical report. Microsoft Corporation, September 1997. http://www.microsoft.com/ie/press/whitepaper/pushwp.htm.

[23] Netscape. *An exploration of dynamic documents*. Netscape Communications Corporation, 1995. http://home.netscape.com/assist/net_sites/pushpull.html.

[24] Netscape. *Netcaster developer's guide*. Netscape Communications Corporation, 1997. http://developer.netscape.com/library/documentation/netcast/devguide/contents.htm.

[25] D. E. Perry and A. L. Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, **17**(4):40–52, October 1992.

[26] PointCast. *Internal PointCast network traffic*. PointCast, August 1997. http://www.pointcast.com/.

[27] PointCast. *Intranet broadcast tools architecture*. PointCast, August 1997. http://www.pointcast.com/.

[28] FTP Software. *World-class push technology from FTP Software*. FTP Software, 1997. http://www.ftp.com/product/whitepapers/push.html.

[29] D. Reed and K. Jones, *Pushing push: advancing the features of channel communication*, *W3C Workshop on Push Technology* (Boston, USA, September 8-9, 1997). W3C, September 1997. http://www.intermind.com/materials/pushing_push.doc.

[30] J. E. White. Mobile Agents. In I. Bradshaw and M. Jeffrey, editors, *Software Agents*. MIT Press and American Association for Artificial Intelligence, 1997.

[31] K. R. Wood, T. Richardon, F. Bennett, A. Harter, and A. Hopper. Global teleporting with Java: towards ubiquitous personalized computing. *IEEE Computer*, **30**(2):53–60, February 1997.