



(22) Date de dépôt/Filing Date: 2007/09/04

(41) Mise à la disp. pub./Open to Public Insp.: 2008/03/07

(45) Date de délivrance/Issue Date: 2012/05/01

(30) Priorité/Priority: 2006/09/07 (EP06120331)

(51) Cl.Int./Int.Cl. *H04L 12/16* (2006.01),
H04L 12/18 (2006.01), *H04L 12/58* (2006.01),
H04W 4/12 (2009.01), *H04W 8/20* (2009.01)

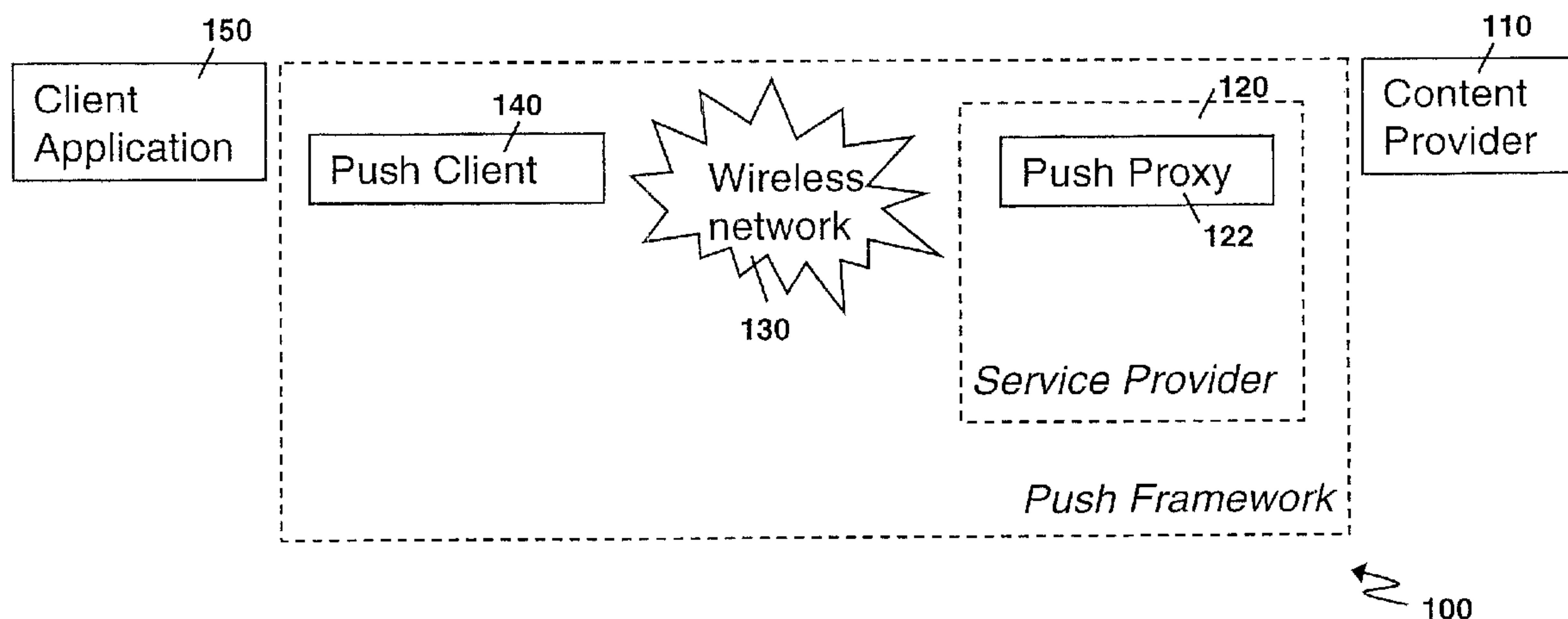
(72) Inventeur/Inventor:
SHENFIELD, MICHAEL, CA

(73) Propriétaire/Owner:
RESEARCH IN MOTION LIMITED, CA

(74) Agent: MOFFAT & CO.

(54) Titre : ENREGISTREMENT PAR ELEMENT INTERMEDIAIRE D'INTEGRATION DES APPLICATIONS DU CLIENT
ET FOURNISSEURS DE CONTENU AVEC SYSTEME DE LIVRAISON DE CONTENU A SOLLICITATION

(54) Title: MEDIATED PLUG-IN REGISTRATION OF CLIENT APPLICATIONS AND CONTENT PROVIDERS WITH
PUSH CONTENT DELIVERY SYSTEM



(57) Abrégé/Abstract:

A method and apparatus for connecting a generic element to a content delivery architecture in a push content delivery system having the steps of: providing a mediator between the generic element and the content delivery architecture; extracting at the mediator metadata for the generic element; and registering the generic element with the content delivery architecture.

ABSTRACT

A method and apparatus for connecting a generic element to a content delivery architecture in a push content delivery system having the steps of: providing a mediator between the generic element and the content delivery architecture; extracting at the mediator metadata for the generic element; and registering the generic element with the content delivery architecture.

MEDIATED PLUG-IN REGISTRATION OF CLIENT APPLICATIONS AND CONTENT PROVIDERS WITH PUSH CONTENT DELIVERY SYSTEM

5 [0001] The present disclosure relates generally to dynamic content delivery in a mobile environment, and in particular to dynamic content delivery utilizing generic applications or generic content providers.

[0002] Users of mobile devices or mobile user equipment (UE) are increasingly becoming more sophisticated in terms of the functionality that they require from their mobile devices and the way that they access data from the mobile devices.

10 [0003] Dynamic content delivery allows users to have information or data pushed to them rather than having to go and seek out the data. Examples of data could include stock quotes, weather updates, traffic updates, dynamic wallpaper, ads, applications or other data desirable to a user.

15 [0004] Current technologies for mobile devices such as wireless application protocol (WAP) have the ability to push content; however, WAP requires websites to be rewritten to satisfy the wireless application protocol and provide users with a uniform site that does not change to accommodate a user's capabilities to view a site.

20 [0005] Other alternatives include SMS based push and broadcast or cell broadcast. In the broadcast case, delivery cannot be customized to the needs of a particular user or the capabilities of a particular device. These systems therefore have no intelligence associated with them. A better solution is required for mobile devices.

25 [0006] In dynamic content delivery, a generic push client serves multiple target applications on a device. Similarly, the push proxy provides services to multiple content providers. It is essential to provide efficient run time registration mechanisms where applications and content providers could register with the dynamic content delivery framework without interrupting service for other applications or content providers. Often the client applications or content providers are generic and don't have intrinsic knowledge of specific content delivery systems.

30 **General**

[0007] The present system and method may provide for a dynamic content delivery architecture and system that allows generic applications and content providers to be added to the system without the necessity to modify the architecture. Specifically, the present system and method allows for a mobile device to become a dynamic application
35 platform in which applications can be added and content provided to the mobile device, where the architecture of the dynamic content delivery system does not limit the type of

application that can be installed on the device nor the type of content that the device receives.

[0008] In one aspect of the present disclosure, metadata may be provided and associated with the content to add intelligence to the content for various processing elements within the dynamic content delivery architecture. This architecture may include logical components that provide for content provision, service provision including push proxies, a wireless network, push client and client applications.

[0009] In a further aspect of the present disclosure, metadata may be provided in a layered "enveloped" model for push content metadata. Content may be wrapped with metadata that can be used for processing at each element within a push framework. The metadata for each successive element may be layered, thereby allowing the processing element to extract only the metadata for that element. For example, a content package that includes metadata directed to a push proxy and a client application can include the content with a first level of metadata for the client application, and a second layer of metadata for the push proxy. Thereby, when the envelope reaches the push proxy, the metadata for the push proxy may be extracted and applied to the content, and the modified content and metadata for the client application is passed to further processing element.

[0010] In another aspect of the present disclosure, the metadata can be split into static metadata (also referred to herein as channel metadata) and dynamic metadata (also referred to herein as content metadata). Static metadata may be established preferably at the time of registration of both the application and the content provider. However, the channel metadata can be established at a later time. The channel metadata may specify processing rules that are specific to the type of content that is being delivered and the application requirements for content type.

[0011] Dynamic metadata may be conversely associated with the specific content being passed.

[0012] In another aspect of the present disclosure, a plug-in registration model may be presented within the push framework. A generic push client and a push proxy are identified, each having various processing blocks or modules that allow these elements to process both content and metadata. These blocks can be directed to process either the content being passed, the metadata being passed or both the content and the metadata being passed.

[0013] Plug-in registration may further provide for the passing of service manifests and application manifests to allow the establishment of channel metadata between a content provider and an application. Specifically, service manifests can be used for registering a

content provider with the push framework, and an application manifest can be used for registering an application with the push framework.

5 [0014] In another aspect of the present disclosure, a method for pushing syndicated content is provided which allows for the handling of data based on its priority and based on network factors including the cost for sending data, the type of network connected to or the users' preferences. An optional mixed push/pull model for syndicated content allows for either a push proxy to push content when network conditions become favorable or for a client to pull content when network conditions become favorable or when the user requires the content.

10 [0015] In order to accommodate various mobile devices, a further aspect of the present disclosure may provide for content fragmentation for content, including non-linear content fragmentation. Non-linear content fragmentation may include augmenting the content with metadata allowing the data to be recomposed once it has been passed to the client.

15 [0016] In a further aspect of the present disclosure, a generic application or content provider could be connected to a specific content delivery enabler (CDE) using a mediated plug in registration model. With mediated plug in registration, the mediator could either be a dedicated application, a generic mediator application using over-the-air application metadata provisioning, or a delivery framework aware run time environment.

20 [0017] These and other aspects will be identified in more detail with respect to the drawings.

[0018] The present application therefore may provide a method for connecting a generic element to a content delivery architecture in a push content delivery system comprising the steps of: providing a mediator between the generic element and the content delivery architecture; extracting at the mediator metadata for the generic element; and registering
25 the generic element with the content delivery architecture.

[0019] A mediator connecting a generic element to a content delivery architecture in a push content delivery system, the mediator comprising: extracting means to extract metadata; and registering means to provide registration information for the generic element to the content delivery architecture.

30

BRIEF DESCRIPTION OF THE DRAWINGS

[0020] The present application will be better understood with reference to the drawings, in which:

35 **Figure 1** is a block diagram of a basic architecture for a dynamic content delivery system;

Figure 2 is a block diagram showing alternative architectures of the dynamic

content delivery system of **Figure 1**;

Figure 3 is the block diagram of **Figure 1** showing content and metadata flow;

Figure 4 is a block diagram showing a push proxy that can be used in association with the present system and method;

5 **Figure 5** is a block diagram showing a push client that can be used in association with the present system and method;

Figure 6 is a block diagram showing a multilayer envelope model of content and metadata;

10 **Figure 7** is the block diagram of **Figure 6**, showing processing steps dynamic metadata for each envelope;

Figure 8 is the block diagram of **Figure 6**, additionally showing processing using static and dynamic metadata;

Figure 9 is a block diagram showing a registration process for an application to a single shared push client;

15 **Figure 10** is a block diagram showing a registration process of an application to a push container managing a pool of push clients;

Figure 11 is a block diagram showing an application registering to a content processor and socket listener;

20 **Figure 12** is a block diagram showing a content provider registering with a single shared push proxy;

Figure 13 is a block diagram showing a content provider registering with a push container managing a pool of push proxies;

Figure 14 is a flow diagram showing registration messages between a content provider and client application;

25 **Figure 15** is a block diagram showing interaction during registration between a push client and push proxy;

Figure 16 is a block diagram showing interaction during registration between a push proxy and a content provider;

30 **Figure 17** is a flow diagram showing the flow of content and metadata between a content provider and processing elements;

Figure 18 is block diagram showing an exemplary transform application for content;

Figure 19 is a block diagram of a content syndication model;

Figure 20 is a block diagram of a linear fragmentation process;

35 **Figure 21** is a block diagram of a non-linear fragmentation process;

Figure 22 is a block diagram of a mediated plug in registration model;

Figure 23 is a block diagram of a application mediator operating in the model of **Figure 22**; and

Figure 24 is a block diagram of an exemplary mobile device that could be used in association with the present method and system.

5

DESCRIPTION OF PREFERRED EMBODIMENTS

[0021] Reference is now made to **Figure 1**. A generic push system for delivering dynamic content to a client application is illustrated. A system of **Figure 1** is a simplified system and shows logical components that need to be in a dynamic content delivery architecture; however, one skilled in the art will appreciate that other components could exist or that various components could be grouped together.

[0022] Architecture 100 includes a content provider 110. Content provider 110 is arranged to provide dynamic content to users that are subscribed with content provider 110. Examples can include, for example, a website selling books. A user may register with content provider 110 to obtain a list of newly released books within specified genres. Other examples could include news sites which might provide headlines to users on a periodic basis, traffic sites which might provide up-to-date traffic information to users during certain periods of the day, stock market sites which could provide updated stock quotes or currency exchange rates to users, among others.

[0023] As will be described in more detail below, content provider 110 registers with a service provider 120 in order to allow clients of the service provider to receive content from content provider 110. Service provider 120 includes a push proxy 122 that acts as a proxy for a client or a client application and provides a destination for content provider 110 to send content.

[0024] Service provider 120 communicates over wireless network 130 with a push client 140 that is located on a mobile device. Push client 140 will be described in more detail below. Push client 140 receives the content that is being delivered from content provider 110 and can communicate the content with a client application 150, which ultimately consumes the content.

[0025] Within the present specification, reference to content provider 110, service provider 120, push proxy 122, wireless network 130, push client 140 or client application 150 is a reference back to the architecture of **Figure 1**.

[0026] Referring to **Figure 2**, it will be appreciated by those skilled in the art that the components of **Figure 1** are merely logical components and are not necessarily separate physical components. **Figure 1** illustrates a generic architecture in which one content provider 110, one push proxy 122, one push client 140 and one client application 150

exist. Alternatives are illustrated in **Figure 2**.

[0027] Specifically, a first alternative architecture **210** includes multiple content providers **110** communicating with a push proxy **122**. Push proxy **122**, as in the architecture of **Figure 1**, communicates over wireless network **130** with a push client **140**. Further, multiple client applications **150** exist in architecture **210**. This is therefore an N-1-1-N system having multiple content providers **110** and multiple client applications **150**.

[0028] Architecture **220** of **Figure 2** includes one content provider **110** communicating with and registered to push proxy **122**. Further, push proxy **122** communicates over wireless network **130** with multiple push clients **140**. Each push client **140** communicates with a client application **150**. Architecture **220** therefore groups the logical components of a client application **150** and a push client **140** and is an N(1-1)-1-1 system.

[0029] Architecture **230** of **Figure 2** has multiple push proxies **122**, each communicating with a content provider **110**. Each push proxy and content provider combination **232** communicates over wireless network **130** with a generic push client **140**, which in turn communicates with client application **150**. This is an 1-1-N(1-1) system.

[0030] In architecture **240** of **Figure 2**, a content provider **110** and push proxy **122** grouping **232** communicates over wireless network **130** with a generic push client **140** and client application **150** combination. This is therefore an N(1-1)-N(1-1) system.

[0031] As will be appreciated by those skilled in the art, other alternatives are possible.

The above shows various logical components, which can be in separate physical components or grouped together. For example, a push client can be imbedded in an application, common shared clients can be used by multiple applications or other alternatives.

[0032] Reference is now made to **Figure 3**. In order to add intelligence to a system, content is associated with a metadata. Metadata, in this case, is defined as data that can be used by a processing element to manipulate the content. As will be appreciated, a generic push system requires metadata to allow various content providers and applications to exist within the system. The metadata can be in various forms, including processing parameters or rules, or a processing handler, code or reference provided directly or a link to a processing handler, code or rules in another location,

[0033] As can be seen in **Figure 3**, content passes from content provider **110** to client application **150** and is illustrated by arrow **310**. Metadata, which provides instructions to various components within the architecture **100** can also pass between components within architecture **100**, usually along with the content. For example, arrow **320** illustrates metadata that originates at the content provider and is transparent to the delivery system until it reaches a client application **150**.

[0034] Arrow 330 shows metadata created by content provider 110 that is intended for the push client 140, and thus only flows to generic push client 140.

[0035] Arrow 340 illustrates metadata generated by service provider 120 and intended for the push client 140, and thus is first associated with the content at the push proxy 122 and stripped from the content at generic push client 140. Examples of where this could occur include agreements between a user and a service provider regarding a billing plan and the level of service to be provided, where the service provider can use the metadata to limit the services available or provide enhanced services.

[0036] The flow of metadata and the role of metadata is described in more detail below.

10 [0037] Reference is now made to **Figure 4**. **Figure 4** illustrates a detailed exemplary push proxy 410 which can be used in association with the present system and method. As will be appreciated by those skilled in the art, push proxy 410 could be the same as push proxy 122 from **Figures 1** and **2**.

15 [0038] Push proxy 410 of **Figure 4** includes various elements that enable push proxy 410 to operate in a generic push environment. This facilitates flexibility since the push proxy is not limited to interaction with specific content providers or push clients, but instead can be adapted to a dynamic environment. The elements described below for push proxy 410 are preferable have within push proxy 410, but the elements are not exhaustive, and other elements are possible. Further, certain elements may be omitted from push proxy 410, with the remaining elements still able to perform generic push services.

20 [0039] Push proxy 410 includes content providers 412 registered to it. Content providers 412 register with a content provider registration service provider interface (SPI) 420. As is described in more detail below, it is desirable in this registration that the content provider 412 includes certain information for the channel being established, referred to herein as channel metadata. Content providers 412 can be the same as content providers 110 of **Figure 1**.

[0040] Push proxy 410 further includes a service administration block 430 to administer the push proxy service.

30 [0041] Push proxy 410 includes various modules to deal with both the content and the metadata associated with that content. A first module is the message broker and delivery queue 440, which is a subsystem that consumes messages from content provider 412 and manages the content delivery queue. As will be appreciated by those skilled in the art, not all content for all client applications can be delivered at once and a delivery queue needs to be established in order to deliver the content in due course. For example, a device may be out of coverage and content may need to be stored.

35

[0042] Push proxy **410** further includes a flow control management block **442**. Flow control management block **442** allows for the control of content flow. For example, a mobile station with limited space may only be able to receive a certain amount of information. In this case, the mobile device, through a push client **140** as illustrated in **Figure 1**, may ask push proxy **410** to stop the flow of data to push client **140**. The flow control management block **442** deals with this.

[0043] Alternatively, the mobile device can be off-line. Flow control management block **442** stops and starts the flow of data to push client **140** when content cannot be delivered as received by push proxy **410**.

[0044] A further component of push proxy **410** is push agents **444**. Push agents **444** are responsible for sending data to clients.

[0045] As will be appreciated by those skilled in the art, blocks **440**, **442** and **444** deal with messaging only, and are not metadata related. In other words, the blocks handle the content of the messages, but not any metadata associated with the content.

[0046] A further component of push proxy **410** is the content metadata extractor and cache block **450**. Content metadata extractor and cache block **450** operate on enveloped content metadata. Specifically, in the envelope model of metadata system, which is described in more detail below, each logical component within the system can have metadata associated with content processing. This metadata allows the logical component to perform actions on the content. Each logical component thus needs to be able to extract the metadata that is associated with it.

[0047] Content metadata extractor and cache block **450** is responsible for extracting metadata that is associated with push proxy **410** and for caching this metadata. The caching function allows optimization by eliminating the need to pass identical metadata in subsequent content envelopes from the same content provider. The extraction and caching of metadata are described below.

[0048] Deferred retrieval message store block **452** is used when it is not efficient to deliver content, or parts of it, to a client application. The deferred retrieval message store block **452** can be used to store content that is not delivered to the client until it is efficient to send the content, or until the content is pulled by the client. The deferred retrieval message store could also be used to cache auxiliary content that could be optionally send to or pulled by the client depending on client application navigation through already delivered content.

[0049] The purpose of deferred retrieval message store block **452** is better explained below with reference to **Figure 19** and **21**. By way of example, deferred retrieval message store block **452** may be used is the case where a user has requested location

information, such as a restaurant close to the location of the user. The content provider or the service provider may have a model of providing information where advertisers can pay to add their information to search requests. Thus, the user that's requesting restaurant information for a location may also have information about stores, golf courses, gyms or other services close to their location attended to their request. A content provider bundles the restaurant information requested with the additional information and passes it to push proxy **410**.

[0050] Push proxy **410** can, based on the metadata provided, create a content package to send to the client. The content package could include the information requested by the client, as well as a digest or summary of related information that the user may be interested in. The summary is sent to the user, but the deferred retrieval message store block **452** stores the actual data that was received from content provider **110**. Thus, if in the future the user wishes to obtain more detailed information about information within the digest, this information is already stored at push proxy **410**.

[0051] An alternative use for deferred retrieval message store block **452** is in the case where a user cannot accept the entire content at once. For example, if it is not feasible or economical to send all content to device, part of the content can be stored until a later time, when it can be pulled by the client or pushed when predefined rules are met. These rules can be specified by the network or service conditions by certain network or service conditions being satisfied. This is described in more detail with reference to **Figure 19** below.

[0052] Push scheduler **454** schedules delivery slots for clients. As described above, in some situations it may not be efficient to push all of the content at once. Push scheduler **452** can determine that it will push some information immediately and the rest according to a predefined schedule. Also, push scheduler **454** may use nature of the content to determine when the content should be pushed. Specifically, metadata may indicate that some content is a high priority or has an expiry that is limited in time, and this content may be pushed immediately, whereas content that has been indicated to have a low priority or with no expiry may be pushed later when conditions for passing data are more favorable.

[0053] As will be appreciated by those skilled in the art, blocks **450**, **452** and **454** deal with both the content of the message and the metadata that is associated with the message.

[0054] Subscription and rules block **460** tracks applications that are registered to receive a service and monitors rules on how to handle particular content being delivered. Content is typically delivered based on a subscription by the client or on behalf of the

client. The user, for example if they want a particular service, can actively request subscriptions. Subscriptions can be made on behalf of a user, for example, if the user has signed an agreement with their service provider **120** to receive a benefit for a service. This could include the case where a user receives a preferred rate as long as the user
5 agrees to receive a certain number of advertisements each day. In this case, the service provider **120** may make the subscription to the advertisement provider on behalf of the client.

[0055] When an application is deleted on a mobile device or when the application unregisters from a subscription, subscription and rules block **460** can unsubscribe that
10 user.

[0056] Content dependencies block **462** is used by push proxy **410** to advertise services that a mobile device user can utilize. Thus, if a mobile device user does not have a screen or bandwidth or memory sufficient for the service, content dependencies block **462** could block the advertisement of that service to the user.

[0057] Content fragmentation block **464** is used to fragment content. This could be used, for example, if the mobile device is unable to receive all of the content at once. Content fragmentation block **464** is used to break the content into various components. It can be used in association with deferred retrieval and message store **452** to store fragmented content that has not yet been delivered.

[0058] Content expiry and replacement block **466** is used for two purposes. First, this block can be used to monitor subscriptions. Each subscription has an expiry time and when this expiry time is met, the subscription can be ended.

[0059] Also, content expiry and replacement block **466** can be used to monitor information. Certain content will have time limits on the validity of the information. For
25 example, a traffic application used to monitor rush hour traffic will be very time dependent. If, for some reason, push proxy **410** is unable to deliver the content immediately to a mobile device, this content is stored in content storage **480** for future delivery. However, if the content is not delivered within a certain specified time period, then it could expire and not be delivered at all.

[0060] Similarly, content replacement deals with a situation where the information is being updated. For example, a client application that is receiving stock quotes may only want the latest stock quote. Thus, if the push proxy **410** is unable to deliver the stock quote to push client **140** and a subsequent stock quote is received from a content provider **110**, metadata within the subsequent stock quote can indicate that it should be
30 used to replace the previous stock quote. Replacement of stored information rather than adding all information to a delivery queue frees space within content storage **480**.

[0061] Channel metadata repository **470** is used to store channel metadata, which is described in more detail below.

[0062] The above describes an exemplary push proxy **410** that can be used with the method and systems herein. The blocks and elements of push proxy **410** allow push proxy **410** to be used in a generic dynamic content delivery system where the type of content and handling of the content at an application can vary and is not predetermined.

[0063] Reference is now made to **Figure 5**. **Figure 5** illustrates a push client **510** that can be used in association with the system and methods herein. Push client **510** can be the same as push client **140** from **Figures 1** and **2**.

[0064] As will be appreciated by those skilled in the art, a push client **510** that is to be used in a generic system in which the content and processing of the content is not predetermined should include blocks or modules that can be used to accommodate both the content and the metadata associated with the content. The blocks defined with regard to **Figure 5** are not meant to be exhaustive, and other blocks could also exist within a push client **510**. Further, the blocks within push client **510** can, in some instances, be omitted without restricting the functionality of the other blocks within push client **510**.

[0065] A push client **510** services applications, and one or more applications **512** can register with push client **510**. The application registration uses an application provider interface **514** as the interface for registration and application provider interface **514** can further be used to extract channel metadata for the application, as described in more detail below.

[0066] Push client **510** includes client administration **520** used to administer the push client **510**.

[0067] As with push server **410** of **Figure 4**, push client **510** includes various blocks that deal with messaging, various blocks that deal with metadata, and various blocks that deal with both messaging and metadata.

[0068] Message broker and application queues **540** handle messages from push proxy **410** for delivery to applications **512**. An application queue is a queue of messages for applications **512**.

[0069] Flow control management block **542** is used to notify push proxy **410** of **Figure 4** to stop pushing content or to resume pushing content. This can be used, for example, when the push client **510** has a limited amount of memory that it can accept pushed content. In this case, before the push content is consumed push client **510** needs to stop the flow of content from push proxy **410**. Once the content has been consumed, flow control management block **542** can be used to start the flow of data again.

[0070] Push agents **544** within push client **510** are used to receive information from push proxy **410** of **Figure 4**.

[0071] As will be appreciated by those skilled in the art, message brokers and application queues **540**, flow control management block **542**, and push agents **544** deal exclusively
5 with messaging and not with metadata.

[0072] Content metadata extractor and cache block **550** is used to extract dynamic metadata destined for push client **510**. As indicated above with reference to push proxy **410** of **Figure 4**, any of the processing elements in the dynamic content delivery architecture could have metadata destined for them and this metadata needs to be
10 extracted. Thus metadata destined for push client **510** is extracted by content metadata extractor and cache block **550**.

[0073] Further, the content metadata extractor and cache block **550** is preferably adapted to cache metadata. Metadata for push client **510** that does not change between a first content package and a second content package does not need to be passed, saving
15 processing time at push client **510** by not requiring the extraction of this metadata, and further saving network resources by not requiring metadata for push client **510** to be passed over wireless network **130**.

[0074] Deferred retrieval manager **552** is used for analyzing fragments of content that are received and putting the content together in the correct way. As described in more detail
20 below, data can be either linear or non-linear. If the data is non-linear, then metadata is required in order to reconstitute it, and this is done by deferred retrieval manager **552**. The deferred retrieval manager **552** also is adapted to analyse a digest of information available in the deferred retrieval store **452** of push proxy **510** and drives the content pull broker **554** (described below) to retrieve this information when required by user. This
25 includes predictive retrieval when content navigation enters a certain branch of the content structure graph or when bandwidth or cost conditions are satisfied

[0075] Content pull broker **554** is used in a push/pull model where the push client **510** is also able to pull content in certain situations. Such situations are described below in more detail with reference to **Figure 19**.

30 [0076] As will be appreciated by those skilled in the art, content metadata extractor and cache **550**, deferred retrieval manager **552** and content pull broker **554** deal both with messaging content and with metadata.

[0077] Subscription management block **560** is the same as subscription and rules block **460** of **Figure 4**. Specifically, subscription management block **560** is used to manage
35 subscriptions. If an application de-registers or is deleted from a mobile device then subscription management block **560** ends the subscription. The subscription

management block **560** can also re-subscribe on behalf of a client application when subscription channel expires.

[0078] Update notification block **562** works with client applications and is used to notify the applications that new content is waiting for them. This can be done in one of three ways:

- a. A first way that update notification block **562** can notify an application **512** is for push client **510** to send the content to application **512** directly.
- b. A second way that update notification block **562** can notify applications **512** of new content is to store the content in content storage **580** and to optionally notify applications **512** that content is waiting. Notification in this case is optional. Specifically, if an application **512** knows that information destined for it is stored within a specific memory block, one option for the application discovering that it has new data is to periodically poll the memory location to see whether there has been something written to it. Alternatively update notification block **562** can send a message to application **512** indicating that it has new data and possibly the location that the data is stored.
- c. A third way that update notification **562** can notify applications **512** of new content is to store the content internally and notify the application. The application can then call on the push client to retrieve the content.

[0079] Content dependency block **564** is the same as content dependency block **462** of **Figure 4**, and can determine whether to advertise the service to the mobile device.

[0080] Content expiry and replacement block **566** is the same as content replacement and expiry block **466** of **Figure 4**. The expiry of content and replacement of content can thus be handled at push client **510** in addition to the push server or push proxy.

[0081] Channel metadata repository **570** is used to store channel metadata for application **512**.

[0082] Background update processing module **575** is used for performing updates when an application **512** is unavailable. The background update allows, for example, the replacement of data with newer data inside the application storage. Thereafter, when a user starts the application, the data displayed by the application is correct and updated.

[0083] Background update processing module **575** uses processing rules translate content into a format acceptable for an application. It can execute and process content in content store **580**.

[0084] By way of example, a task list that is updated for a contractor overnight could have tasks pushed to it. The task application is not started during this time, and background update processing module 575 can be used to update the content for the task application. This could be done with code for handling an extensible mark-up language (XML) file, and could exist on the device in a file called "handler.exe".
5 Background update processing block 575 on push client 510 can run handler.exe, passing the XML document as a parameter. The handler then constructs the task into the application's internal format.

[0085] Once the background update processing block 575 of push client 510 constructs
10 the task into the application internal format, it then can read the task into the task list from content storage 580 and append the new task to the list. It then can store the modified back to content storage 580 for when the task application next connects to push client 510.

[0086] Figure 5 therefore illustrates a push client 510 that can be used in a generic
15 dynamic content delivery system, where content and processing of the content is dynamic and not predetermined. The blocks described above with reference to the push client 510 of Figure 5 are used to accommodate the dynamic nature of the system.

[0087] As indicated above with reference to Figure 3, content is associated with metadata to provide intelligence for the processing of the content. In accordance with the
20 present method and system, metadata can be divided into two types of metadata. Specifically, static (channel) metadata and dynamic (content) metadata.

[0088] Due to the unlimited possibilities of types of content providers and applications, metadata is critical in order to build generic systems. The only way to handle the specific type of content is through metadata.

[0089] Static metadata is metadata that provides rules on how to process specific types
25 of content. Static metadata can be broken into various levels of abstraction and include for example structural information about the content itself. For example, a Real-time Simple Syndication (RSS) document could be delivered with an RSS 2.0.XSD structure, and all content from that content provider will be delivered with this structure.

[0090] A further level of abstraction for static metadata includes the provision of
30 processing rules for content subtype. This could be application specific. Thus, for example, a financial news application indicates that data should be extracted from a financial news RSS stream, stored in a predefined location, and that the application should be notified about the arrival of the information. The application always requires
35 content destined for it to be handled in this way.

[0091] The static metadata (also referred to herein as channel metadata) stays the same

throughout the subscription between the application and the content provider, and thus the static metadata can be established once for each element within the architecture and for each content delivery channel. In one embodiment this is done at the time of registration of the application or the content provider.

5 **[0092]** Dynamic metadata is metadata that is associated with a particular piece of content. For example, expiry information associated with a particular piece of data or replacement rules and information associated with a particular piece of data (i.e. document K replaces document L).

10 **[0093]** As indicated above with reference to **Figures 4** and **5**, each processing entity can receive both static and dynamic metadata that is directed at that processing entity. Thus push proxy **410** uses the content metadata extractor and cache **450** to extract the dynamic metadata, and content expiry and replacement block **466** is used to replace undelivered content with newer content received at push proxy **410**.

15 **[0094]** Reference is now made to **Figure 6**. **Figure 6** illustrates a multilayer envelope model for content metadata.

20 **[0095]** A push proxy **410** receives a push envelope **610** that includes content processing metadata for the proxy server **612** and a push client envelope **614**. The push proxy **410** extracts content processing metadata **612** and uses this metadata to process push client envelope **614**. Metadata **612** dictates to push proxy what to do with the push client envelope **614**.

25 **[0096]** Push client envelope **614** is passed to push client **510** where it is broken into a content envelope **620** and a content processing metadata **622**. Content processing metadata **622** is used by push client **510** to process the content envelope **620**. For example, this can be used to instruct push client **510** to perform replacement of previously delivered content envelope **620** with the latest envelope if client application **150** is only interested in the latest version of the content.

30 **[0097]** Content envelope **620** is passed to client application **150**. Content envelope **620** includes content processing metadata **630** for the application and the content payload **632** that is to be consumed by client application **150**.

35 **[0098]** As will be appreciated by those skilled in the art, the nesting of envelopes in accordance with **Figure 6** provides for a rich dynamic environment in which processing can occur at any processing element of the architecture and which the content provider **110** can specify how specific content is to be dealt with. In one embodiment, metadata directed to a particular logical element is opaque to other processing elements.

40 **[0099]** Alternatively, the service provider **120** can also add metadata at push proxy **410** for processing at push client **510** or client application **150**.

[00100] Referring to **Figure 7**, this figure shows the envelope model of **Figure 6** and the steps that each processing element takes with an envelope. As illustrated in **Figure 7**, push proxy **410** first extracts the metadata from push envelope **610**. This is done in step **710**.

5 [00101] In step **712**, push proxy **410** uses the metadata to process the push client envelope **614**. In step **714**, push proxy **410** delivers the push client envelope **614** to push client **510**.

[00102] Similarly, push client **510**, in step **720** extracts the content processing metadata **622** from push client envelope **614**. In step **722**, push client **510** uses the content
10 processing metadata **622** on content envelope **620**. In step **724**, the push client **510** delivers content envelope **620** to client application **150**.

[00103] In step **730**, client application **150** extracts the content processing metadata **630** and in step **732** uses the content processing metadata **630** on content payload **632**.

[00104] Referring to **Figure 8**, this figure shows the method as illustrated in **Figure 7** with
15 the additional step of the use of static or channel metadata. Specifically, after the metadata has been extracted in step **710** from push envelope **610**, the push proxy **410** next uses the static channel metadata to process the push client envelope in step **810**. In step **712**, push proxy **410** next processes the content processing dynamic metadata **612**. Push proxy **410** next delivers the push client envelope **614** in step **714**.

20 [00105] Similarly, push client **510** extracts the content processing metadata **622** in step **720**. Push client **510** then uses the channel metadata in step **820** on the content within content envelope **620**. Push client **510** then, in step **722**, uses the dynamic content metadata in content processing metadata **622** prior to delivering content envelope **620** to client application **150** in step **724**.

25 [00106] Client application **150** first extracts, in step **730**, content processing metadata **630**. It then uses the channel metadata in step **830** on content payload **632**. Client application **150** then uses, in step **732**, content processing metadata **630** on content payload **632**.

[00107] As will be appreciated by those skilled in the art, the above model therefore
30 allows for both static metadata to be applied for the channel along with dynamic metadata that is associated with the particular content being sent.

[00108] Reference is now made to **Figure 9**. As will be appreciated from **Figure 5**, push client **510** can serve multiple target applications **512** on a mobile device. An efficient runtime registration mechanism is required where applications can register with the
35 dynamic content delivery framework without interrupting service for other applications.

[00109] Referring to **Figure 9**, push client **510** includes three applications, specifically

applications **910**, **912** and **914** that are already registered with the push client. As will be appreciated, the plug in model is important because new devices can allow unlimited application types to be installed on the device. Further, applications can be installed dynamically, leading to a mobile device becoming an application platform. Because the device can be an application platform, it must be capable of dynamically incorporating new applications.

[00110] Push proxy **410** further includes a service administration block **430** to administer the push proxy service.

[00111] Push proxy **410** includes various modules to deal with both the content and the metadata associated with that content. A first module is the message broker and delivery queue **440**, which is a subsystem that consumes messages from content provider **412** and manages the content delivery queue. As will be appreciated by those skilled in the art, not all content for all client applications can be delivered at once and a delivery queue needs to be established in order to deliver the content in due course. For example, a device may be out of coverage and content may need to be stored.

[00112] Push proxy **410** further includes a flow control management block **442**. Flow control management block **442** allows for the control of content flow. For example, a mobile station with limited space may only be able to receive a certain amount of information. In this case, the mobile device, through a push client **140** as illustrated in **Figure 1**, may ask push proxy **410** to stop the flow of data to push client **140**. The flow control management block **442** deals with this.

[00113] Alternatively, the mobile device can be off-line. Flow control management block **442** stops and starts the flow of data to push client **140** when content cannot be delivered as received by push proxy **410**.

[00114] A further component of push proxy **410** is push agents **444**. Push agents **444** are responsible for sending data to clients.

[00115] As will be appreciated by those skilled in the art, blocks **440**, **442** and **444** deal with messaging only, and are not metadata related. In other words, the blocks handle the content of the messages, but not any metadata associated with the content.

[00116] A further component of push proxy **410** is the content metadata extractor and cache block **450**. Content metadata extractor and cache block **450** operate on enveloped content metadata. Specifically, in the envelope model of metadata system, which is described in more detail below, each logical component within the system can have metadata associated with content processing. This metadata allows the logical component to perform actions on the content. Each logical component thus needs to be able to extract the metadata that is associated with it.

[00117] Content metadata extractor and cache block **450** is responsible for extracting metadata that is associated with push proxy **410** and for caching this metadata. The caching function allows optimization by eliminating the need to pass identical metadata in subsequent content envelopes from the same content provider. The
5 extraction and caching of metadata are described below.

[00118] Deferred retrieval message store block **452** is used when it is not efficient to deliver content, or parts of it, to a client application. The deferred retrieval message store block **452** can be used to store content that is not delivered to the client until it is efficient to send the content, or until the content is pulled by the client. The deferred
10 retrieval message store could also be used to cache auxiliary content that could be optionally send to or pulled by the client depending on client application navigation through already delivered content.

[00119] The purpose of deferred retrieval message store block **452** is better explained below with reference to **Figure 19** and **21**. By way of example, deferred
15 retrieval message store block **452** may be used is the case where a user has requested location information, such as a restaurant close to the location of the user. The content provider or the service provider may have a model of providing information where advertisers can pay to add their information to search requests. Thus, the user that's requesting restaurant information for a location may also have information about stores,
20 golf courses, gyms or other services close to their location attended to their request. A content provider bundles the restaurant information requested with the additional information and passes it to push proxy **410**.

[00120] Push proxy **410** can, based on the metadata provided, create a content package to send to the client. The content package could include the information
25 requested by the client, as well as a digest or summary of related information that the user may be interested in. The summary is sent to the user, but the deferred retrieval message store block **452** stores the actual data that was received from content provider **110**. Thus, if in the future the user wishes to obtain more detailed information about information within the digest, this information is already stored at push proxy **410**.

[00121] An alternative use for deferred retrieval message store block **452** is in the
30 case where a user cannot accept the entire content at once. For example, if it is not feasible or economical to send all content to device, part of the content can be stored until a later time, when it can be pulled by the client or pushed when predefined rules are met. These rules can be specified by the network or service conditions by certain
35 network or service conditions being satisfied. This is described in more detail with reference to **Figure 19** below.

- [00122]** Push scheduler **454** schedules delivery slots for clients. As described above, in some situations it may not be efficient to push all of the content at once. Push scheduler **452** can determine that it will push some information immediately and the rest according to a predefined schedule. Also, push scheduler **454** may use nature of the content to determine when the content should be pushed. Specifically, metadata may indicate that some content is a high priority or has an expiry that is limited in time, and this content may be pushed immediately, whereas content that has been indicated to have a low priority or with no expiry may be pushed later when conditions for passing data are more favorable.
- [00123]** As will be appreciated by those skilled in the art, blocks **450**, **452** and **454** deal with both the content of the message and the metadata that is associated with the message.
- [00124]** Subscription and rules block **460** tracks applications that are registered to receive a service and monitors rules on how to handle particular content being delivered. Content is typically delivered based on a subscription by the client or on behalf of the client. The user, for example if they want a particular service, can actively request subscriptions. Subscriptions can be made on behalf of a user, for example, if the user has signed an agreement with their service provider **120** to receive a benefit for a service. This could include the case where a user receives a preferred rate as long as the user agrees to receive a certain number of advertisements each day. In this case, the service provider **120** may make the subscription to the advertisement provider on behalf of the client.
- [00125]** When an application is deleted on a mobile device or when the application unregisters from a subscription, subscription and rules block **460** can unsubscribe that user.
- [00126]** Content dependencies block **462** is used by push proxy **410** to advertise services that a mobile device user can utilize. Thus, if a mobile device user does not have a screen or bandwidth or memory sufficient for the service, content dependencies block **462** could block the advertisement of that service to the user.
- [00127]** Content fragmentation block **464** is used to fragment content. This could be used, for example, if the mobile device is unable to receive all of the content at once. Content fragmentation block **464** is used to break the content into various components. It can be used in association with deferred retrieval and message store **452** to store fragmented content that has not yet been delivered.
- [00128]** Content expiry and replacement block **466** is used for two purposes. First, this block can be used to monitor subscriptions. Each subscription has an expiry time and

when this expiry time is met, the subscription can be ended.

[00129] Also, content expiry and replacement block **466** can be used to monitor information. Certain content will have time limits on the validity of the information. For example, a traffic application used to monitor rush hour traffic will be very time dependent. If, for some reason, push proxy **410** is unable to deliver the content immediately to a mobile device, this content is stored in content storage **480** for future delivery. However, if the content is not delivered within a certain specified time period, then it could expire and not be delivered at all.

[00130] Similarly, content replacement deals with a situation where the information is being updated. For example, a client application that is receiving stock quotes may only want the latest stock quote. Thus, if the push proxy **410** is unable to deliver the stock quote to push client **140** and a subsequent stock quote is received from a content provider **110**, metadata within the subsequent stock quote can indicate that it should be used to replace the previous stock quote. Replacement of stored information rather than adding all information to a delivery queue frees space within content storage **480**.

[00131] Channel metadata repository **470** is used to store channel metadata, which is described in more detail below.

[00132] The above describes an exemplary push proxy **410** that can be used with the method and systems herein. The blocks and elements of push proxy **410** allow push proxy **410** to be used in a generic dynamic content delivery system where the type of content and handling of the content at an application can vary and is not predetermined.

[00133] Reference is now made to **Figure 5**. **Figure 5** illustrates a push client **510** that can be used in association with the system and methods herein. Push client **510** can be the same as push client **140** from **Figures 1** and **2**.

[00134] As will be appreciated by those skilled in the art, a push client **510** that is to be used in a generic system in which the content and processing of the content is not predetermined should include blocks or modules that can be used to accommodate both the content and the metadata associated with the content. The blocks defined with regard to **Figure 5** are not meant to be exhaustive, and other blocks could also exist within a push client **510**. Further, the blocks within push client **510** can, in some instances, be omitted without restricting the functionality of the other blocks within push client **510**.

[00135] A push client **510** services applications, and one or more applications **512** can register with push client **510**. The application registration uses an application provider interface **514** as the interface for registration and application provider interface **514** can further be used to extract channel metadata for the application, as described in more

detail below.

[00136] Push client **510** includes client administration **520** used to administer the push client **510**.

5 **[00137]** As with push server **410** of **Figure 4**, push client **510** includes various blocks that deal with messaging, various blocks that deal with metadata, and various blocks that deal with both messaging and metadata.

[00138] Message broker and application queues **540** handle messages from push proxy **410** for delivery to applications **512**. An application queue is a queue of messages for applications **512**.

10 **[00139]** Flow control management block **542** is used to notify push proxy **410** of **Figure 4** to stop pushing content or to resume pushing content. This can be used, for example, when the push client **510** has a limited amount of memory that it can accept pushed content. In this case, before the push content is consumed push client **510** needs to stop the flow of content from push proxy **410**. Once the content has been consumed, flow
15 control management block **542** can be used to start the flow of data again.

[00140] Push agents **544** within push client **510** are used to receive information from push proxy **410** of **Figure 4**.

20 **[00141]** As will be appreciated by those skilled in the art, message brokers and application queues **540**, flow control management block **542**, and push agents **544** deal exclusively with messaging and not with metadata.

[00142] Content metadata extractor and cache block **550** is used to extract dynamic metadata destined for push client **510**. As indicated above with reference to push proxy **410** of **Figure 4**, any of the processing elements in the dynamic content delivery architecture could have metadata destined for them and this metadata needs to be
25 extracted. Thus metadata destined for push client **510** is extracted by content metadata extractor and cache block **550**.

[00143] Further, the content metadata extractor and cache block **550** is preferably adapted to cache metadata. Metadata for push client **510** that does not change between a first content package and a second content package does not need to be passed,
30 saving processing time at push client **510** by not requiring the extraction of this metadata, and further saving network resources by not requiring metadata for push client **510** to be passed over wireless network **130**.

[00144] Deferred retrieval manager **552** is used for analyzing fragments of content that are received and putting the content together in the correct way. As described in more
35 detail below, data can be either linear or non-linear. If the data is non-linear, then metadata is required in order to reconstitute it, and this is done by deferred retrieval

manager **552**. The deferred retrieval manager **552** also is adapted to analyse a digest of information available in the deferred retrieval store **452** of push proxy **510** and drives the content pull broker **554** (described below) to retrieve this information when required by user. This includes predictive retrieval when content navigation enters a certain branch of the content structure graph or when bandwidth or cost conditions are satisfied

[00145] Content pull broker **554** is used in a push/pull model where the push client **510** is also able to pull content in certain situations. Such situations are described below in more detail with reference to **Figure 19**.

[00146] As will be appreciated by those skilled in the art, content metadata extractor and cache **550**, deferred retrieval manager **552** and content pull broker **554** deal both with messaging content and with metadata.

[00147] Subscription management block **560** is the same as subscription and rules block **460** of **Figure 4**. Specifically, subscription management block **560** is used to manage subscriptions. If an application de-registers or is deleted from a mobile device then subscription management block **560** ends the subscription. The subscription management block **560** can also re-subscribe on behalf of a client application when subscription channel expires.

[00148] Update notification block **562** works with client applications and is used to notify the applications that new content is waiting for them. This can be done in one of three ways:

- a. A first way that update notification block **562** can notify an application **512** is for push client **510** to send the content to application **512** directly.
- b. A second way that update notification block **562** can notify applications **512** of new content is to store the content in content storage **580** and to optionally notify applications **512** that content is waiting. Notification in this case is optional. Specifically, if an application **512** knows that information destined for it is stored within a specific memory block, one option for the application discovering that it has new data is to periodically poll the memory location to see whether there has been something written to it. Alternatively update notification block **562** can send a message to application **512** indicating that it has new data and possibly the location that the data is stored.
- c. A third way that update notification **562** can notify applications **512** of new content is to store the content internally and notify the application. The application can then call on the push client to retrieve the content.

[00149] Content dependency block **564** is the same as content dependency block **462** of **Figure 4**, and can determine whether to advertise the service to the mobile device.

[00150] Content expiry and replacement block **566** is the same as content replacement and expiry block **466** of **Figure 4**. The expiry of content and replacement of
5 content can thus be handled at push client **510** in addition to the push server or push proxy.

[00151] Channel metadata repository **570** is used to store channel metadata for application **512**.

[00152] Background update processing module **575** is used for performing updates
10 when an application **512** is unavailable. The background update allows, for example, the replacement of data with newer data inside the application storage. Thereafter, when a user starts the application, the data displayed by the application is correct and updated.

[00153] Background update processing module **575** uses processing rules
15 translate content into a format acceptable for an application. It can execute and process content in content store **580**.

[00154] By way of example, a task list that is updated for a contractor overnight
could have tasks pushed to it. The task application is not started during this time, and background update processing module **575** can be used to update the content for the task application. This could be done with code for handling an extensible mark-up
20 language (XML) file, and could exist on the device in a file called "handler.exe". Background update processing block **575** on push client **510** can run handler.exe, passing the XML document as a parameter. The handler then constructs the task into the application's internal format.

[00155] Once the background update processing block **575** of push client **510**
25 constructs the task into the application internal format, it then can read the task into the task list from content storage **580** and append the new task to the list. It then can store the modified back to content storage **580** for when the task application next connects to push client **510**.

[00156] **Figure 5** therefore illustrates a push client **510** that can be used in a
30 generic dynamic content delivery system, where content and processing of the content is dynamic and not predetermined. The blocks described above with reference to the push client **510** of **Figure 5** are used to accommodate the dynamic nature of the system.

[00157] As indicated above with reference to **Figure 3**, content is associated with metadata to provide intelligence for the processing of the content. In accordance with the
35 present method and system, metadata can be divided into two types of metadata. Specifically, static (channel) metadata and dynamic (content) metadata.

[00158] Due to the unlimited possibilities of types of content providers and applications,

metadata is critical in order to build generic systems. The only way to handle the specific type of content is through metadata.

[00159] Static metadata is metadata that provides rules on how to process specific types of content. Static metadata can be broken into various levels of abstraction and include for example structural information about the content itself. For example, a Real-time Simple Syndication (RSS) document could be delivered with an RSS 2.0.XSD structure, and all content from that content provider will be delivered with this structure.

[00160] A further level of abstraction for static metadata includes the provision of processing rules for content subtype. This could be application specific. Thus, for example, a financial news application indicates that data should be extracted from a financial news RSS stream, stored in a predefined location, and that the application should be notified about the arrival of the information. The application always requires content destined for it to be handled in this way.

[00161] The static metadata (also referred to herein as channel metadata) stays the same throughout the subscription between the application and the content provider, and thus the static metadata can be established once for each element within the architecture and for each content delivery channel. In one embodiment this is done at the time of registration of the application or the content provider.

[00162] Dynamic metadata is metadata that is associated with a particular piece of content. For example, expiry information associated with a particular piece of data or replacement rules and information associated with a particular piece of data (i.e. document K replaces document L).

[00163] As indicated above with reference to **Figures 4 and 5**, each processing entity can receive both static and dynamic metadata that is directed at that processing entity. Thus push proxy **410** uses the content metadata extractor and cache **450** to extract the dynamic metadata, and content expiry and replacement block **466** is used to replace undelivered content with newer content received at push proxy **410**.

[00164] Reference is now made to **Figure 6**. **Figure 6** illustrates a multilayer envelope model for content metadata.

[00165] A push proxy **410** receives a push envelope **610** that includes content processing metadata for the proxy server **612** and a push client envelope **614**. The push proxy **410** extracts content processing metadata **612** and uses this metadata to process push client envelope **614**. Metadata **612** dictates to push proxy what to do with the push client envelope **614**.

[00166] Push client envelope **614** is passed to push client **510** where it is broken into a content envelope **620** and a content processing metadata **622**. Content processing

metadata **622** is used by push client **510** to process the content envelope **620**. For example, this can be used to instruct push client **510** to perform replacement of previously delivered content envelope **620** with the latest envelope if client application **150** is only interested in the latest version of the content.

5 **[00167]**Content envelope **620** is passed to client application **150**. Content envelope **620** includes content processing metadata **630** for the application and the content payload **632** that is to be consumed by client application **150**.

[00168]As will be appreciated by those skilled in the art, the nesting of envelopes in accordance with **Figure 6** provides for a rich dynamic environment in which processing
10 can occur at any processing element of the architecture and which the content provider **110** can specify how specific content is to be dealt with. In one embodiment, metadata directed to a particular logical element is opaque to other processing elements.

[00169]Alternatively, the service provider **120** can also add metadata at push proxy **410** for processing at push client **510** or client application **150**.

15 **[00170]**Referring to **Figure 7**, this figure shows the envelope model of **Figure 6** and the steps that each processing element takes with an envelope. As illustrated in **Figure 7**, push proxy **410** first extracts the metadata from push envelope **610**. This is done in step **710**.

[00171]In step **712**, push proxy **410** uses the metadata to process the push client envelope **614**. In step **714**, push proxy **410** delivers the push client envelope **614** to push
20 client **510**.

[00172]Similarly, push client **510**, in step **720** extracts the content processing metadata **622** from push client envelope **614**. In step **722**, push client **510** uses the content processing metadata **622** on content envelope **620**. In step **724**, the push client **510**
25 delivers content envelope **620** to client application **150**.

[00173]In step **730**, client application **150** extracts the content processing metadata **630** and in step **732** uses the content processing metadata **630** on content payload **632**.

[00174]Referring to **Figure 8**, this figure shows the method as illustrated in **Figure 7** with the additional step of the use of static or channel metadata. Specifically, after the
30 metadata has been extracted in step **710** from push envelope **610**, the push proxy **410** next uses the static channel metadata to process the push client envelope in step **810**. In step **712**, push proxy **410** next processes the content processing dynamic metadata **612**. Push proxy **410** next delivers the push client envelope **614** in step **714**.

[00175]Similarly, push client **510** extracts the content processing metadata **622** in step
35 **720**. Push client **510** then uses the channel metadata in step **820** on the content within content envelope **620**. Push client **510** then, in step **722**, uses the dynamic content

metadata in content processing metatadata 622 prior to delivering content envelope 620 to client application 150 in step 724.

[00176] Client application 150 first extracts, in step 730, content processing metadata 630. It then uses the channel metadata in step 830 on content payload 632. Client application 150 then uses, in step 732, content processing metadata 630 on content payload 632.

[00177] As will be appreciated by those skilled in the art, the above model therefore allows for both static metadata to be applied for the channel along with dynamic metadata that is associated with the particular content being sent.

10 [00178] Reference is now made to Figure 9. As will be appreciated from Figure 5, push client 510 can serve multiple target applications 512 on a mobile device. An efficient runtime registration mechanism is required where applications can register with the dynamic content delivery framework without interrupting service for other applications.

[00179] Referring to Figure 9, push client 510 includes three applications, specifically applications 910, 912 and 914 that are already registered with the push client. As will be appreciated, the plug in model is important because new devices can allow unlimited application types to be installed on the device. Further, applications can be installed dynamically, leading to a mobile device becoming an application platform. Because the device can be an application platform, it must be capable of dynamically incorporating new applications.

[00180] As seen in Figure 9, application 916 wants to register with push client 510. Application 916 includes an application manifest 918 that, in a preferred embodiment, provides the channel metadata for the application. Specifically, application manifest 918 provides information to push client 510, and ultimately push proxy 410 and content provider 110 from Figure 1 with the static metadata for the application. This can include, but is not limited to, what type of content the application expects, how the content will be delivered, whether the application needs notification, or other channel information that would be evident to those skilled in the art having regard to the present system and method.

30 [00181] Application 916 therefore registers with push client 510, providing application manifest 918 to establish a channel to a content provider for servicing application 916.

[00182] Referring to Figure 10, an alternate model could be the model described with regard to architecture 220 of Figure 2. Specifically, in the model of Figure 10, a client application 150 is paired with a push client 140. Each of the client application 150/push client 140 pairs are coordinated with a push container 1010.

[00183] When application 1020 wishes to register with push container 1010, a client 140

is created, or if it already exists is used, by push container **1010**. Further, in registration, the application **1020** provides an application manifest **1030** to push container **1010**, thereby providing channel metadata (static metadata) for application **1020**.

5 **[00184]** An alternative illustration of **Figure 10** is shown in **Figure 11**. Specifically, a push container **1110** manages/maintains a pool of push clients. When an application registers with the container it obtains a dedicated push client **510**, which in the simple case could be represented by a pair of a socket listener **1130** and content handler. The push client is returned to the pool when the application unregisters from the container (and content delivery service) or is deleted from the device.

10 **[00185]** Push container **1110** includes sockets **1120** for communication. Further, push container **1110** includes socket listeners **1130** and content processors **1140** assigned to a particular socket.

[00186] As seen in **Figure 11**, various content processor and socket listener pairs are used by previously registered applications **150**.

15 **[00187]** When a new application **1150** wants to register with push container **1110**, a new content processor and socket listener **1120** and **1130** are assigned to service application **1050**.

20 **[00188]** The above therefore provides for a generic push framework in which a client application **150** that is new can be implemented and registered with a push client **510** or push container **1010** or **1110**, thereby allowing the device to become an application platform capable of dynamically incorporating new applications. The passing of an application manifest **1030** or **918** from **Figures 9** and **10** above allows for the establishment of channel metadata, thereby allowing the content to be processed according to the application's requirements.

25 **[00189]** Referring to **Figure 12**, content providers **110** similarly need to register with a push proxy **410**. As seen in **Figure 12**, push proxy **410** includes three content providers, namely, **1210**, **1212** and **1214**, already registered with push proxy **410**. Content provider **1216** desires to register with push proxy **410**.

30 **[00190]** Similarly to the application manifest **918** illustrated in **Figure 9** provided by an application **916** when registering with push client **510**, content provider **1216** includes a service manifest **1218** that is passed to push proxy **410** when content provider **1216** registers. Service manifest **1218** includes information concerning the type of information that the content provider will provide, how often it provides this information, the format of the information, and any other information that is useful for the service or for
35 advertisement of the service. Other information is possible.

[00191] Push proxy **410** thus uses service manifest **1218** to establish channel (static)

metadata for content provider **1216**.

[00192] Referring to **Figure 13**, an alternative embodiment, represented by architecture **230** of **Figure 2**, is to have a push container with a number of push proxy **122** and content provider **110** pairings. As with **Figure 12**, various applications could already be registered with push container **1310**, and in the example of **Figure 12**, applications **1312**, **1314** and **1316** are already registered with push proxies **1313**, **1315** and **1317** respectively.

[00193] A new application **1320** wants to register with push container **1310**. Thus, push container **1310** creates a new proxy (not shown) or uses an existing proxy (not shown) with which it associates content provider **1320**. Further, content provider **1320** provides service manifest **1322** to describe the content that content provider **1320** will be providing, thereby allowing the establishment of channel metadata.

[00194] As will be appreciated by those skilled in the art, the embodiments of **Figures 9** and **10** show two options for push clients, either with shared applications or with dedicated push clients per application. One skilled in the art will realize that other embodiments are possible. Similarly, with respect to **Figures 12** and **13**, a push proxy with multiple content providers registered to it is shown or a dedicated push proxy for each content provider, and embodied in a push container is shown.

[00195] With reference to **Figure 14**, messaging between a content provider **110** and a client application **150** is shown. Content provider **110** provides a registration message to push proxy **410**. This message can include the service manifest which can be used to provide channel metadata to push proxy **410**. This is done in step **1410**.

[00196] Content provider **110** may also or alternatively provide channel metadata in a subsequent message, as illustrated by step **1412**.

[00197] Push proxy **410** then adds a service to a list of available services (the service catalogue) in step **1414**.

[00198] An optional step in the example of **Figure 14** is for push proxy **410** to notify push client **510** of the new service available in step **1416** and this notification may be propagated to a client application **110** in step **1418**.

[00199] As will be appreciated by those skilled in the art, steps **1416** and **1418** are optional, and other alternatives include client application **150** pulling the service catalogue periodically from push proxy **410** to view new services.

[00200] When a user or service provider for client application **150** decides that client application **150** should subscribe to a service, it sends a subscription message in step **1420**. The subscription message is further passed to push proxy **410** in step **1422**.

[00201] Once push proxy **410** receives the subscription message in step **1422**, two

options are available. A first option is to send a message **1424** to content provider **110** for a subscription and then receive a message envelope that includes metadata back in step **1426**. The metadata could be device or device type specific and is propagated 1428/1432 towards the client application 150.

5 **[00202]** Alternatively, push proxy **410** may receive the subscription message in step **1422** and immediately, based on information already provided by content provider **110** and stored on push proxy **410** reply in step **1430** to push client **510**. This reply is propagated to the client application **150** in step **1432**. As will be appreciated, the reply can include channel metadata specific for content provider **110**.

10 **[00203]** The difference in models can be dependent on who is customizing the data for the application. As will be appreciated, content provider **110** provides the best customization of content compared with other processing elements. However, service provider **120**, through push proxy **410**, can also provide for customization of content.

[00204] Further, as will be appreciated, the structure of the content could be dependent on the data that the application requires. For example, in a financial application, the application may want both stock quotes and currency rates. The following XML may be used:

```

20      <FIN>
          <quotes>
              <quote ticker = ABC>
                  18.54
              </quote>
              <quote ticker = XYZ>
                  123.45
25              </quote>
          </quotes>
          <rates>
              <rate id = "US-CAN">
                  1.15
30              </rate>
              <rate id = "US-EURO">
                  0.85
              </rate>
          </rates>
35      </FIN>

```

[00205] If the user only wanted quotes and no currency exchange, the structure could change to:

```

40      <FIN>
          <quote ticker = ABC>
              18.54
          </quote>
          <quote ticker = XYZ>
              123.45
45          </quote>
      </FIN>

```

[00206]The metadata can provide information to the application on the structure that of the data being passed.

[00207]Thus, two models exist. Static metadata can be provided to push proxy 410 and to push client 510 either during registration or afterwards. Alternatively, the metadata for push proxy 410 and push client 510 can be pre-provisioned, i.e. information is stored at a push client or a push proxy until an application registers with a client.

[00208]Reference is now made to **Figure 15**. **Figure 15** shows logical steps that occur upon registration of an application with a push client 510.

10 [00209]Once an application registers with push client 510, a first step 1510 is to match the registered application with the content type required by the application. This is known from the application manifest 918 as illustrated in **Figure 9**.

[00210]A second step 1520 is to set up the environment for the application. These include but are not limited to storage and delivery options for the application. For example, an application may limit transmissions to a predetermined amount of data. The push client 510 in a flow control event, or if the application or client is out of touch, may require the caching of the data for the application and optionally to notify the application that data is waiting.

[00211]A third step 1530, is to notify push proxy 410 of the application settings. This includes for example available storage for the application or push client 510. As will be appreciated, push proxy 410 should not push more data than push client 510 can store. Thus, the application settings could include an upper limit of the data that is passed. Referring to **Figures 4 and 5**, this could invoke content fragmentation block 464 to fragment the content if it is greater than the application can process. Also, if the data is non-linear, content dependencies block 462 may be required to create metadata for content dependencies block 564 of **Figure 5** in order to allow content dependencies block 564 to reconstitute the data.

[00212]Referring again to **Figure 15**, step 1530 can also indicate preference on data delivery. For example, the application may prefer certain types of data over others and these types of data may be given priority. Thus step 1530 can be used to establish a delivery schedule where data of type "A" is delivered immediately while data of type "B" can be delivered at a deferred time.

[00213]Reference is now made to **Figure 16**. When a content provider 110 registers with a push proxy 410, various steps are performed. A first step 1610 includes analyzing required client settings for content storage and delivery. This can be used, for example, for service advertisement in order to identify push clients 510 on devices capable of

consuming content from content provider **110**.

[00214] A second step **1620** allows push proxy **410** to set up the environment, including proxy storage, delivery options, transformation options, among others.

[00215] In step **1630**, push proxy **410** can check whether the application is already
5 registered to obtain content from a content provider **110**. If this is the case, the application is ready to receive content and a notification from push proxy **410** to content provider **110** that the delivery channel is established and the application is ready for content can be sent.

[00216] Step **1630** can occur, for example, if an application is pre-installed on a device
10 prior to content provider **110** coming on-line. Thus, the application is waiting for content provider **110** to become available or the application is of generic type (e.g. a browser or RSS Viewer) and is capable of consuming information from multiple content providers. In an alternative setting, if content provider **110** is already available before the application is installed, the notification step **1530** in **Figure 15** can be used to initiate the content
15 starting to flow from content provider **110** to a client application **150**.

[00217] As will be appreciated with reference to **Figure 16**, client settings can include certain information such as the available storage size used for content partitioning, the queue size used for flow control, delivery scheduling including a push interval, whether the client is retrieving information from the proxy, creating a pseudo-push mode,
20 customization options such as the screen size of a mobile device, among others.

[00218] As will be further appreciated, service catalogues may differ for different clients. For example, certain clients may be able to utilize more data, have a different screen size or other conditions which make the client more suitable for a content provider **110** than a device that cannot handle this amount of information, has a smaller screen size, etc.
25 Thus, push proxy **410** can create a service catalogue for specific client applications based on knowledge of those client applications, and only those devices with that client application **150** installed can receive information concerning the content provider.

[00219] As will be further appreciated, in some cases the application may be installed based on a service provider and content provider without the user intervention. For
30 example, if content provider **110** registers with push proxy **410**, a user of a mobile device may have a contract obligation to accept a certain application. Thus push proxy **410** could notify push client **510** that it is ready to install an application and push the application to push client **510**. This could, for example, include a user that has agreed to receive a certain number of ads each month in order to get a preferred rate on their
35 mobile plan. The content provider **110** could be an ad provider and push proxy **410** may therefore push an advertisement displaying application to push client **510**, which might be

served by an application installer registered with push client 410, thereby having the content provider 110 and the service provider 120 entirely driving the process.

[00220]The above therefore provides for a plug-in registration model in a push framework where each application or content provider registers and provides an application manifest or service manifest respectively. The application manifest or service manifest is used to establish channel metadata at the push proxy 410 and push client 510 either during registration or subsequently. Thereafter, when an application 150 registers and a content provider 110 registers, content can start flowing between the application 150 and the content provider 110.

10 [00221]With reference to Figures 4 and 5, the channel metadata is stored in a channel metadata repository 470 and 570. It is, however, also advantageous to store dynamic metadata on the various processing elements within architecture 100 if the dynamic metadata is repeated. As will be appreciated, this will save processing on the push proxy 410 since current metadata extractor 450 does not need to extract the same metadata over and over. Further, processing by various modules such as content expiry and replacement module 466 or 566 do not need to be updated for each piece of content that is passed. Since push proxy 410 could be working with a large number of push clients 510, this processing saving for each content message could be significant. Further, bandwidth could be saved by not having to pass the metadata over a fixed line between content provider 110 and push proxy 410 or over the air between push proxy 410 and push client 510.

[00222]Reference is now made to Figure 17. Figure 17 illustrates an example of run time flow where your last metadata version is stored by the processing element.

25 [00223]As seen in Figure 17, content provider 110 provides a content envelope which includes content $[C_1 + M(p, c, a)_1]$. This means that a first content payload is being sent along with metadata that includes proxy metadata, client metadata and application metadata. This is sent in step 1710.

30 [00224]At step 1712, push proxy 410 uses the proxy metadata as illustrated by the phrase "use $M(p)_1$ ". Further, in step 1714 the content plus the metadata that includes the client metadata and the application metadata is passed to push client 510.

[00225]In step 1716, push client 510 uses the client metadata and further in step 1718, passes the content payload to client application 150. Client application 150 uses, in step 1720 the application metadata and further consumes the content payload.

35 [00226]As seen in step 1722, a second content payload, designated by C_2 , has the same metadata as the first content payload. Because each processing element, namely, push proxy 410, push client 510 and client application 150, cached the metadata for content

provider **110**, the metadata does not need to be passed again but instead already resides on the processing element.

[00227] Thereafter, in step **1724** the push proxy **410** uses metadata that was previously cached for the push proxy **410**. Similarly, in steps **1726** and **1728** the push client **510** uses the client metadata and the client application **150** uses the application metadata respectively. Content is passed, without metadata, in steps **1725** and **1727**.

[00228] As illustrated in step **1740**, content may have new metadata for the push client **510** and client application **150**, but may keep the old metadata for the push proxy **410**. In this case, the metadata that is passed in step **1740** includes only client metadata and application metadata. In step **1742**, the push proxy **410** uses the cached proxy metadata and passes the content payload along with the new client metadata and application metadata in step **1744**.

[00229] In step **1746**, the push client **510** uses the new client metadata that was passed to it and further passes the content payload and application metadata in step **1748**.

[00230] In step **1750**, the client application uses the new application metadata and further consumes the content payload.

[00231] As will be appreciated by one skilled in the art, various configurations could exist concerning which metadata has changed and which metadata stays the same, and only the metadata that has changed is passed to the processing element that requires it. As will be appreciated by those skilled in the art, the processing element, if it does not receive new metadata, goes back to the cached metadata that it has stored and uses this on the content payload.

[00232] In a further alternative embodiment, incremental changes can also be made to metadata. For example, in step **1760** a new content payload along with a delta metadata version can be passed to service proxy **410**. The delta of the proxy metadata can include a difference between the proxy metadata previously passed and the current metadata that the content should be processed with. The push proxy **410** composes the metadata by adding the previous metadata with the delta and then using this to process the content payload in step **1762**. Thereafter, since there has been no change, in step **1764** the content payload is sent by itself and in step **1766** the push client **510** uses the previously cached client metadata.

[00233] Push client then passes the content payload in step **1768** to client application **150**, which uses the previously cached location metadata on the content payload in step **1770** and then it consumes the content payload.

[00234] An example of where incremental data may be used is a situation in which a content provider tells the proxy that of the existent fields within the content payload, 30

should be extracted to send to client application **150**. In a subsequent transaction, two additional fields that are important for that piece of content payload may be deemed necessary to be passed to the client application **150** by content provider **110**. The content provider could therefore, using an incremental change, tell push proxy to extract
5 the two additional fields and add them to the 30 fields that were previously extracted. By only having to pass the delta, i.e. the two additional fields, the processing time for extracting the metadata at push proxy **410** is reduced, thereby optimizing the process.

[00235]As will be further appreciated, metadata can come in various forms. It could be compiled such as native code or interpreted code such as Java or C#. The metadata can
10 also be a data/properties file that indicates to use certain properties. In another alternative embodiment, it can be binary content, for example a transformation such as a XSLT transformation on an XML document.

[00236]The above can be used for various applications to provide intelligence for content being transferred to a specific client application. It can also provide for rich content
15 providers that can provide content for various applications merely based on the metadata that they provide with their data. This can be illustrated by way of example in **Figure 18**.

[00237]A content provider **110** could, for example, be a on-line bookseller. An application can register with the on-line bookseller to indicate to the on-line bookseller that it wants to be informed of new releases of a specific genre. This could occur on a
20 daily or weekly or monthly basis.

[00238]Content provider **110**, for example, on a weekly basis will send a content envelope **1810** having a book list **1812**, to push proxy **410**. It can also send a transform metadata **1814**, which can be, for example, a URL link for transforming the specific content based on the application receiving it.

25 [00239]In one embodiment, the book list **1812** could include numerous books, descriptions of each book including the author and a synopsis of the book. The file may, for example, be 100 KB in size.

[00240]Push proxy **410** can receive this large file and may realize, based on the client application being serviced, that a transformation to the large content file needs to be done
30 in order to better accommodate the client which may only be able to receive, for example, 10 kilobytes of information. The transformation that is passed as a proxy metadata can therefore be applied to the book list to reduce the book list to a 10 KB modified document **1820**. This can, for example, be done by removing the synopsis, ranking the books and only including the top 50 or other transformations as would be evident to those skilled in
35 the art.

[00241]Once the transformation is complete, the modified document **1820** is then sent to

the push client **510**.

[00242] Further, the deferred retrieval message store **452**, as seen in **Figure 4**, can be used to store the extra content that was stripped out in the transformation process.

[00243] The advantage of the above is that the bookseller can have one site and send one list to all of its clients. Since various clients will not be mobile wireless clients, the 100 KB file may be appropriate for these clients. By also providing the transformation metadata, the bookseller can have one list that it sends to everyone. As will be appreciated by those skilled in the art, most current web technologies require a separate website for a mobile client, and this is overcome by the above solution.

[00244] The above also lends itself to a syndication model and reference is now made to **Figure 19**.

[00245] As will be appreciated by those skilled in the art, a mobile device may not wish to receive large amounts of data when network conditions are not optimal for the receiving of large amounts of data. Further, network operators may wish to avoid sending large amounts of data during peak periods of bandwidth usage in order to spread network traffic more evenly over time. This can be accomplished using a push/pull model as illustrated in **Figure 19**.

[00246] As described with reference to **Figure 4** above, content may be provided that includes more information than the user may currently need. For example, if the user requests location information for restaurants within his area, a service provider may wish to add advertising such as other services available in the area. However, the service provider may not wish to push this additional content immediately to the user, but instead provide a primer such as a headline or a table of contents showing the additional content.

[00247] In other situations, the content may be too large to send to the user, and the user may receive only the first part of the content and the remainder of the content is stored in a deferred retrieval message store **452**.

[00248] Thereafter, the stored content can be passed to push client **510** either by push proxy **410** or when asked for by push client **510**.

[00249] Push client **510** includes a network status monitor **1910** which can monitor the status of the network. Push client **510** may wish to only receive extra data in certain conditions. For example, on a hybrid mobile device that has a WiFi and a cellular option, it is cheaper to provide data on the WiFi connection, and thus network status monitor **1910** could wait until the push client **510** is connected to a WiFi network prior to getting the deferred content. Alternatively, network status monitor could check whether the client is roaming in a foreign network or connected to the home network in order to minimize roaming charges. Network status monitor may also check to see whether a dedicated

data channel is established for the device. One skilled in the art will realize that network status monitor **1910** could also check for various other preconditions in the network before requesting deferred data to be passed to push client **510**.

5 **[00250]** A wireless network **130** could also provide information to either or both of push client **510** and push proxy **410** concerning the costs of delivery of data. As will be appreciated by those skilled in the art, various peak periods occur for the delivery of content. In the case of traffic information, the peak periods may be at the beginning and end of the workday when people are coming to and going from work. For stock quotes the peak period may be during the time that the market is open. Other peak periods will
10 exist. In order to average the data traffic, it may be desirable for the network to charge different rates based on the current data usage in the network. Thus during peak periods a higher rate may be charged than a non-peak period such as the middle of the night. Wireless network **130** therefore provides delivery cost notifications to a deferred retrieval manager **552** on a push client **510** and to push scheduler **454** on push proxy **410**.

15 **[00251]** In one embodiment, data from content provider **110** and passed to push proxy **410** can be ranked based on its importance to the client. Certain information can be designated through metadata to be delivered immediately. Other information can be designated to be delivered when the network cost is less than a first value (for example 10¢ per megabyte) and other data may be designated to be delivered when the network
20 costs drop below a second value (for example, 5¢ per megabyte). Thus push scheduler **454** considers the data that is stored in deferred retrieval message store **452** and instructs push agent **444** to pass deferred data to push agent **544** on push client **510**.

[00252] Alternatively, deferred retrieval manager **552** could also monitor network conditions as sent from wireless network **130** and if the data rate is below a certain rate
25 can ask content pull broker **554** to pull content from deferred retrieval message store **452**.

[00253] Alternatively, deferred retrieval manager **552** could see that the network status is favorable for pulling larger amounts of data, such as if the mobile device has connected with a WiFi network, and ask content pull broker **554** to pull the data from deferred retrieval message store **452**.

30 **[00254]** As will be further appreciated, a user can always request to have the content pulled. Thus user request **1940** could also be used to trigger content pull broker **554** to pull the data from deferred retrieval message store **452**.

[00255] The rules stored in push scheduler **454** and deferred retrieval manager **552** could be static metadata based on a classification of content. The rules could also be based on
35 dynamic metadata for the particular data that has been passed. In this case the content provider **110** has classified the data.

[00256] Reference is now made to **Figure 20**. As will be appreciated by those skilled in the art, data can be one of two forms, linear or non-linear. Linear data could, for example, be arrays or strings or content that flows in a linear fashion. Non-linear data, conversely, is data that does not linearly relate to each other and can include complex dependencies with content maps or links.

[00257] For linear content, fragmentation merely involves the breaking of the data into various components based on linear progression. The data is partitioned into segments and the segments are delivered to the push client **410**. As indicated in **Figure 20**, fragmentation processor **2010** interacts with content **2012** and decides that the content can be parsed with linear progression. The fragmentation processor **2010** next partitions the data into segments **2014**, **2016** and **2018** in the example of **Figure 20**, and, as illustrated in **Figure 20**, passes the first segment **2014** while deferring the passing of the second and third segments **2016** and **2018** respectively.

[00258] The cursor management module **2030** keeps track of which segment has been delivered and delivers the next segment in order.

[00259] Referring to **Figure 21**, non-linear content needs to be partitioned in a more intelligent way. Further, at the other end, in order to reconstitute the segments, metadata is required.

[00260] A fragmentation processor **2110** analyses the content based on a metadata based analysis. These could include keeping certain segments or data elements together if logically required. Fragmentation processor **2110** analyses content **2112** and partitions the content into segments based on logical rules. Each segment includes the content plus metadata including for example, dependencies, maps, and navigation rules for each segment.

[00261] Once partitioned, a first segment **2114** is sent to push client **510** and the passing of the remainder of the segments **2116** and **2118** is deferred as illustrated in **Figure 21**. Segment navigation block **2130** deals with which segment to send next. As will be appreciated by those skilled in the art, first segment **2114** includes a data portion and a metadata portion. The metadata portion of segment **2114** is a layer of metadata that is added by the fragmentation processor **2110** to indicate to content dependencies module **564** how to reconstitute the content. Data portion of first segment **2114** can include both content and metadata associated with the channel or with the content.

[00262] Segment navigation block **2130** is adapted to process how a user travels through the data. For example, if the data is in a tree format and the user goes down a first branch of the tree, segment navigation block **2130** may pass to push client **410** other branches in the tree that can be reached from the element that the user has navigated to.

[00263] For example, a tree could include an employee database that has employee names along with a structure for the corporation. Based on **Figure 21**, if the user navigates into a specific department of the organization, the segmentation navigation block **2130** might forward the group fragments for groups within that department. If the user then navigates into a specific group within the department, the segmentation navigation block **2130** might then pass information fragments about the employees within that group.

[00264] The above therefore requires that the data be partitioned into logical components. Identifiers are assigned to all types and content, and structural information is created passing the information with the primer.

[00265] The above therefore provides an architecture for dynamic content delivery that can be used with generic systems where applications and content can be added without changing the structure of the system. The content can be tailored to fit the application receiving it, and be fragmented according to the above.

[00266] While the above provides for generic systems in which the content providers and applications are aware of the content delivery environment, in one embodiment of the method and system of the present disclosure either a content provider, application, or both, are not aware of the content delivery environment. Reference is now made to **Figure 22**. In **Figure 22**, a content delivery enabler (CDE) **2210** is adapted to facilitate dynamic content delivery between a content provider **2230** and a client application and the ability to bind the content provider **2230** with an application. Content delivery enabler **2210** is equivalent to push framework **100** of **Figure 1**.

[00267] Content delivery enabler **2210** includes a push client **2212**, a wireless network **2214** and a push server **2216**. These can be equated with push client **140**, wireless network **130** and service provider **120** of **Figure 1** respectively, and the descriptions of the components with relation to **Figure 1** are applicable to the embodiment of **Figure 22**.

[00268] Content provider **2230** is, for the embodiment of **Figure 22**, a generic content provider. In other words, content provider **2230** is used by different terminals utilizing different delivery means and is therefore agnostic to any particular content delivery architecture. Such generic purpose content providers **2230** would be known to those skilled in the art and serve a broad variety of clients.

[00269] A user of a mobile device could, in some circumstances, access content provider **2230** through regular browsing. However, if more sophisticated services such as dynamic content delivery are required from a general purpose content provider **2230**, means are required in order to bring the generic content provider **2230** into the content delivery architecture as described above. The content provider **2230** must be registered

with the content delivery enabler **2210** in order to ensure that any application that would like to receive content from content provider **2230** will be given an option of registering with the content provider **2230**. Further, content delivery enabler **2210** will need to have metadata information about the content being provided from generic content provider **2230**, in order to establish the delivery schema and schedule, identify applications or application types capable of processing the content, or any other information needed from a content delivery enabler **2210** and an application perspective.

[00270] In order to bring content provider **2230** into a content delivery architecture as described above with reference to **Figures 1 to 21**, metadata will need to exist for the content provider **2230**.

[00271] Similarly, a client application such as client application **2240**, **2245** or **2250** as illustrated in **Figure 22**, may be a generic client application, referred to herein as generic application **2255**. In other words, the generic application **2255** could be agnostic to the content delivery architecture.

[00272] When a generic application **2255** is unaware of the content delivery model and not specifically designed to interact to receive dynamic content through a content delivery model, such applications **2255** will be unable on their own to register with a content delivery enabler **2210**. These applications **2255** typically have no knowledge of the content delivery enabler and of registration interfaces and thus do not fit in to the above model as described with reference to **Figures 1 to 21**.

[00273] As with a generic content provider **2230**, it is desirable to engage generic applications **2255**. Such applications provide a user of a mobile device with more choice concerning the applications loaded onto the mobile device and the functionality of the application once loaded.

[00274] In order to engage an application such as generic application **2255**, the present disclosure provides a mediator **2260**. The role of mediator **2260** is to have knowledge about the metadata of the applications it registers and about the registration interface including metadata schema for registering the application.

[00275] Mediator **2260** can be either generic or application specific. For example, mediator **2260** can be provided by an application developer along with the application. Mediator **2260** is then run, for example, when the application is loaded onto a mobile device in order to register the particular application with the particular content delivery enabler. In this case, mediator **2260** includes metadata information for the application.

[00276] Alternatively, a generic mediator **2260** could exist on the mobile device. In that case, once application registration is required, for example by having an application loaded onto the mobile device, having a user request registration of the application with a

content delivery enabler **2210**, receiving push data destined for an application, among other triggers, the generic mediator **2260** could go to a repository **2280** and obtain metadata for the application.

5 **[00277]** As will be appreciated once metadata is received by a mediator **2260** for an application **2255**, the mediator will then be required to bind the application with the content delivery enabler **2210**.

10 **[00278]** Reference is now made to **Figure 23**. A mediator **2370** attempts to resolve metadata **2320** that is associated with an application **2310** to metadata schema **2340** required by the content delivery enabler **2345**. Specifically, an application **2310** includes metadata associated with it, called herein, application metadata **2320**. Application metadata **2320** includes various parameters that the application **2310** is expecting. Examples include storage parameters, including where content that is arriving is stored, fragmentation or size limitations of the data that the application is willing to accept, among others. Other parameters include, for example, notification parameters
15 including whether the application should be notified of content arrival or if content should be provided directly to the application such as by using a cache, or if the content should be stored in a predefined location. Other metadata specific for an application is also contemplated to be within the scope of the present disclosure.

20 **[00279]** As further seen in **Figure 23**, content delivery enabler **2345** includes metadata schema **2340** associated therewith, referred to herein as content delivery enabler metadata schema **2340**. As will be appreciated, content delivery enabler metadata schema **2340** deals with application related parameters that are needed by content delivery enabler **2345** to be able to handle and deliver content for the application.

25 **[00280]** Mediator **2370** resolves application metadata **2320** with content delivery enabler metadata schema **2340**. Such resolution includes the steps of reading application metadata **2320**, extracting needed parameters and information, adding delivery handling related information to application metadata **2320**, converting data into a format acceptable for application **2310**, and proceeding to register application **2310** with the content delivery enabler **2345** including push client **2348**. The adding of delivery
30 handling related information could be based on pre-specified parameters of the device or could include user input, such as if the user is prompted to specify information about data that they wish to receive or the handling of that information.

35 **[00281]** Similarly, a mediator **2270** from **Figure 22** performs similar functions between metadata for a content provider **2230** and push server **2216**. Mediator **2270** connects parameters between the content provider metadata and the content delivery enabler metadata.

[00282] Mediators **2260** and **2270** thereby allow for generic applications and generic content providers respectively. In order have mediator **2260** available to perform the above functionality various options are available.

5 [00283] The run time environment on a mobile device will either be content delivery aware or not content delivery aware. If the run time environment is content delivery aware, then the run time environment could act in one of two ways. In the first way, the run time environment could see that a new application is installed, the user wishes to connect an application to a content delivery system, or other trigger that indicates that a generic application **2255** is to be connected to a content delivery enabler **2210**.

10 [00284] In one embodiment the runtime environment could then initiate an application specific mediator **2260** being downloaded from a central repository. Mediator **2260** can be provisioned over the air at various times, including when an application is first loaded onto a mobile device, when the user requests dynamic content delivery from the application, when content arrives that could be serviced by the mobile device, among
15 others. An application specific mediator **2260** could be provided from a service provider managed repository of application mediators or could be provided by an application developer or retailer in a repository for applications provided by the application developer or retailer.

[00285] Once an application specific mediator **2260** is downloaded, the runtime
20 environment can execute the downloaded mediator **2260**, thereby binding the application to the content delivery enabler.

[00286] Alternatively, the run time environment itself could act as a mediator application. In this case, the environment knows how to execute registration process and could download metadata for an application from a central repository. At this point, the run time
25 environment could use the metadata to register the application with the content delivery enabler **2210**.

[00287] In a further embodiment, if the run time environment is not content delivery aware, a user will need to initiate the loading of a special application for generic application registration. The special application makes the device content delivery aware
30 and the downloaded special application could then, whenever a generic application registration trigger occurs, proceed to find the metadata for the generic application, and register the application using the metadata found.

[00288] The above therefore describes that the loading, executing and binding of an application and metadata associated therewith could be either automatic or user driven.
35 In other words, the user could initiate the process or the process could be automatic whenever an application is downloaded on to the device. Three models are disclosed –

the loading of an application specific mediator from a repository by a content delivery aware runtime environment; the loading of application metadata from a repository by a content delivery aware runtime environment that acts as a generic mediator; or the loading of a special application onto the mobile device to make a non-content delivery aware runtime environment aware of the content delivery architecture, whereupon the now content delivery aware architecture could download metadata and act as a generic mediator.

[00289] A further option available is the pushing of registration metadata or mediator to the mobile device that has installed a particular application. The pushing could occur from the service provider or from a third party entity such as an application repository, for example, upon the downloading of the application from the repository. Further, when a user installs a push client **2212** onto the mobile device, thereby joining content delivery framework, the service provider could at that time push registration metadata for applications already installed on the mobile device. The list of applications could be provided by the device in this case. Other options will be apparent to those skilled in the art having regards to the present disclosure.

[00290] As indicated above, an alternative to a central repository for getting either the metadata or the mediator application itself is to provide a repository with an application. In other words, various repositories would exist for each application developer. Each application developer would then be able to provide a mediator or metadata that is specific for the application. In a preferred embodiment, the metadata or mediator could also be specific for the content delivery architecture that is being utilized. In this way, mediator **2260** from **Figure 22** or **2370** from **Figure 23** would not need to perform a matching of parameters since the parameters between the content delivery enabler metadata **2340** and the application metadata **2320** would match.

[00291] For content providers **2230**, mediation is performed on the push server **2216** and is analogous to the above. In other words, push server **2216** can obtain a mediator for a specific content provider or include a generic mediator that can get metadata for a content provider. The mediation occurs between content delivery enabler metadata **2340** and content provider metadata (not shown).

[00292] The above can be further expanded by considering implicit and explicit binding. As will be appreciated by those skilled in the art having regard to the above disclosure, an application is bound to a content provider. In other words, a content provider can indicate the application or application type that it expects will handle data provided from the content provider.

[00293] In one example, a content provider can specify that only a specific application can

handle content from it. The content delivery enabler **2210** will then only notify a mobile device that has the specific application installed that the content provider is available.

The metadata and parameters for the application are known from the content provider, and the binding of the application to the content provider is therefore explicit.

5 **[00294]** The above content provider and application could be generic. In other words, the content provider and application could be agnostic to the content delivery architecture. The above could therefore require a mediator between both the content provider and the content delivery enabler **2210**, and between the content delivery enabler **2210** and the application **2255**.

10 **[00295]** The application **2255** can also be configured to provide explicit binding. For example, the application can specify that content is required from a specific URI for the content provider **2230**. If the application **2255** is then registered prior to the content provider **2230**, the mobile device will be notified as soon as the content provider **2230** requested by the application **2255** registers.

15 **[00296]** The relations in explicit binding are managed on the server side. The content provider enabler **2210** has information about the devices and applications **2255** that are connected to the content delivery enabler **2210** and will only send notifications to those devices that have application **2255** present.

20 **[00297]** Alternatively, implicit binding can occur. For example, a content provider **2230** may specify that it is providing a certain type of content. If an application **2230** specifies that it can process all or a subset of that type of content, it may be considered to be acceptable for content provider **2230**.

25 **[00298]** In an implicit binding situation, both the content provider **2230** and the application **2255** must provide a content type token to the content delivery enabler **2210**. The content delivery enabler **2210** must then match whether the schema for the token of a content provider links with the token for the application.

[00299] In one example, a content provider **2230** indicates that it provides an RSS feed. If an application indicates that it can process RSS feeds, then a match exists and the implicit binding can occur.

30 **[00300]** As will be appreciated, content providers may provide numerous types of data and an application may not be able to process all of those types of data. Additionally, the content provider may provide specialized or customized information and the application may only offer generic processing. For example, if a stock quotes RSS content provider provides a specialized RSS feed, for stock quote information, a generic RSS Feed Viewer
35 application may not provide expected user experience.

[00301] In one embodiment, a strength metric could be used to decide whether an

application should be tied to a content provider. Thus, if an application does not include any of the schema provided by a content provider, a strength of zero could be registered and the application would not be tied to the content provider.

5 [00302] The strength could relate to the number of information types provided by the content provider **2230** that can be interpreted by the application **2255**. Thus an integer from 0 to X, where X can be any integer, could be the strength metric.

[00303] The strength metric can be used to make decision about binding applications to content providers. For example, if a strength of one existed then the user may wish to bind the application to the content provider and a prompt could be provided to the user. If
10 a strength of two existed, for example, then the binding could occur automatically. The above is only meant as an example and various tying schemes could be implemented.

[00304] With the example above, if the content provider therefore provided a content token that contains "RSS", "financial", and "stock quote" segments, and the generic RSS Feed Viewer application provided content token "RSS", the strength would be defined as
15 one. If, on the other hand, the application would be designed as customized RSS Viewer for financial information and provided content token with "RSS" and "financial" segments, the strength could be two.

[00305] As will be appreciated by those skilled in the art with reference to the above disclosure, the tying of an application to a content provider is important to the content
20 delivery enabler **2210** architecture. The mobile device is notified when new content providers are added for which applications can process their information. Further, when an application is registering with the content delivery enabler **2210**, the application **2255** could be provided with a list of content providers **2230** that the application could utilize. Since not all applications **2255** or content providers **2230** will explicitly provide for which
25 providers **2230** or applications **2255** respectfully they are intended to be used with, the implicit tying of an application to a content provider needs to be performed by the content delivery enabler **2210**.

[00306] The strength metric described above is only one example of a strength metric that could be used. Further, the schema for the content type token is also generic, but as will
30 be appreciated needs to be common for both the application and content provider. The content delivery enabler **2210** must apply some heuristics to identify candidates for implicit binding. Further, the content of the token can be opaque to the content delivery enabler **2210** and in some cases the schema of the token could also be opaque. This, for example, exists where binding occurs for an exact match of the content type.

35 [00307] Another example of a strength metric is a ratio, which could be used instead of an absolute integer. Thus, if the content provider **2230** provides three segments in a content

token (e.g. "RSS", "financial", "stock quotes") then the ratio of a generic RSS Viewer application (e.g. "RSS") is 1/3 and of specialized RSS Viewer for financial information (e.g. "RSS" and "financial" segments) is 2/3.

5 [00308] The above therefore provides for the registering of generic applications and generic content providers with a content delivery enabler architecture, and further provides for the explicit or implicit tying of applications to content providers depending on the application and the content provider.

[00309] As will be appreciated, the push client and client applications can reside on any mobile device. One exemplary mobile device is described below with reference to **Figure**
10 **24**. This is not meant to be limiting, but is provided for illustrative purposes.

[00310] **Figure 24** is a block diagram illustrating a mobile station apt to be used with preferred embodiments of the apparatus and method of the present application. Mobile station **2400** is preferably a two-way wireless communication device having at least voice and data communication capabilities. Mobile station **2400** preferably has the capability to
15 communicate with other computer systems on the Internet. Depending on the exact functionality provided, the wireless device may be referred to as a data messaging device, a two-way pager, a wireless e-mail device, a cellular telephone with data messaging capabilities, a wireless Internet appliance, or a data communication device, as examples.

20 [00311] Where mobile station **2400** is enabled for two-way communication, it will incorporate a communication subsystem **2411**, including both a receiver **2412** and a transmitter **2414**, as well as associated components such as one or more, preferably embedded or internal, antenna elements **2416** and **2418**, local oscillators (LOs) **2413**, and a processing module such as a digital signal processor (DSP) **2420**. As will be
25 apparent to those skilled in the field of communications, the particular design of the communication subsystem **2411** will be dependent upon the communication network in which the device is intended to operate.

[00312] Network access requirements will also vary depending upon the type of network **2419**. In some CDMA networks network access is associated with a subscriber or user of mobile station **2400**. A CDMA mobile station may require a removable user identity
30 module (RUIM) or a subscriber identity module (SIM) card in order to operate on a CDMA network. . The SIM/RUIM interface **2444** is normally similar to a card-slot into which a SIM/RUIM card can be inserted and ejected like a diskette or PCMCIA card. The SIM/RUIM card can have approximately 64K of memory and hold many key configuration
35 **2451**, and other information **2453** such as identification, and subscriber related information.

[00313] When required network registration or activation procedures have been completed, mobile station 2400 may send and receive communication signals over the network 2419. As illustrated in Figure 24, network 2419 can consist of multiple base stations communicating with the mobile device. For example, in a hybrid CDMA 1x EVDO system, a CDMA base station and an EVDO base station communicate with the mobile station and the mobile station is connected to both simultaneously. The EVDO and CDMA 1x base stations use different paging slots to communicate with the mobile device.

[00314] Signals received by antenna 2416 through communication network 2419 are input to receiver 2412, which may perform such common receiver functions as signal amplification, frequency down conversion, filtering, channel selection and the like, and in the example system shown in Figure 24, analog to digital (A/D) conversion. A/D conversion of a received signal allows more complex communication functions such as demodulation and decoding to be performed in the DSP 2420. In a similar manner, signals to be transmitted are processed, including modulation and encoding for example, by DSP 2420 and input to transmitter 2214 for digital to analog conversion, frequency up conversion, filtering, amplification and transmission over the communication network 2419 via antenna 2418. DSP 2420 not only processes communication signals, but also provides for receiver and transmitter control. For example, the gains applied to communication signals in receiver 2412 and transmitter 2414 may be adaptively controlled through automatic gain control algorithms implemented in DSP 2420.

[00315] Mobile station 2400 preferably includes a microprocessor 2438 which controls the overall operation of the device. Communication functions, including at least data and voice communications, are performed through communication subsystem 2211. Microprocessor 2438 also interacts with further device subsystems such as the display 2422, flash memory 2424, random access memory (RAM) 2426, auxiliary input/output (I/O) subsystems 2428, serial port 2430, two or more keyboards or keypads 2432, speaker 2434, microphone 2436, other communication subsystem 2440 such as a short-range communications subsystem and any other device subsystems generally designated as 2442. Serial port 2430 could include a USB port or other port known to those in the art.

[00316] Some of the subsystems shown in Figure 24 perform communication-related functions, whereas other subsystems may provide "resident" or on-device functions. Notably, some subsystems, such as keyboard 2432 and display 2422, for example, may be used for both communication-related functions, such as entering a text message for transmission over a communication network, and device-resident functions such as a

calculator or task list.

[00317] Operating system software used by the microprocessor **2438** is preferably stored in a persistent store such as flash memory **2424**, which may instead be a read-only memory (ROM) or similar storage element (not shown). Those skilled in the art will
5 appreciate that the operating system, specific device applications, or parts thereof, may be temporarily loaded into a volatile memory such as RAM **2426**. Received communication signals may also be stored in RAM **2426**.

[00318] As shown, flash memory **2424** can be segregated into different areas for both computer programs **2458** and program data storage **2450**, **2452**, **2454** and **2456**. These
10 different storage types indicate that each program can allocate a portion of flash memory **2424** for their own data storage requirements. Microprocessor **2438**, in addition to its operating system functions, preferably enables execution of software applications on the mobile station. A predetermined set of applications that control basic operations, including at least data and voice communication applications for example, will normally
15 be installed on mobile station **2400** during manufacturing. Other applications could be installed subsequently or dynamically.

[00319] A preferred software application may be a personal information manager (PIM) application having the ability to organize and manage data items relating to the user of the mobile station such as, but not limited to, e-mail, calendar events, voice mails,
20 appointments, and task items. Naturally, one or more memory stores would be available on the mobile station to facilitate storage of PIM data items. Such PIM application would preferably have the ability to send and receive data items, via the wireless network **2419**. In a preferred embodiment, the PIM data items are seamlessly integrated, synchronized and updated, via the wireless network **2419**, with the mobile station user's corresponding
25 data items stored or associated with a host computer system. Further applications may also be loaded onto the mobile station **2400** through the network **2419**, an auxiliary I/O subsystem **2428**, serial port **2430**, short-range communications subsystem **2440** or any other suitable subsystem **2442**, and installed by a user in the RAM **2426** or preferably a non-volatile store (not shown) for execution by the microprocessor **2438**. Such flexibility
30 in application installation increases the functionality of the device and may provide enhanced on-device functions, communication-related functions, or both. For example, secure communication applications may enable electronic commerce functions and other such financial transactions to be performed using the mobile station **2400**.

[00320] In a data communication mode, a received signal such as a text message or web
35 page download will be processed by the communication subsystem **2411** and input to the microprocessor **2438**, which preferably further processes the received signal for output to

the display **2422**, or alternatively to an auxiliary I/O device **2428**. A push client **2460**, which could be equivalent to push clients **140**, **510** or **2212**, could also process the input.

[00321] A user of mobile station **2400** may also compose data items such as email messages for example, using the keyboard **2432**, which is preferably a complete alphanumeric keyboard or telephone-type keypad, in conjunction with the display **2422** and possibly an auxiliary I/O device **2428**. Such composed items may then be transmitted over a communication network through the communication subsystem **2411**.

[00322] For voice communications, overall operation of mobile station **2400** is similar, except that received signals would preferably be output to a speaker **2434** and signals for transmission would be generated by a microphone **2436**. Alternative voice or audio I/O subsystems, such as a voice message recording subsystem, may also be implemented on mobile station **2400**. Although voice or audio signal output is preferably accomplished primarily through the speaker **2434**, display **2422** may also be used to provide an indication of the identity of a calling party, the duration of a voice call, or other voice call related information for example.

[00323] Serial port **2430** in **Figure 24**, would normally be implemented in a personal digital assistant (PDA)-type mobile station for which synchronization with a user's desktop computer (not shown) may be desirable, but is an optional device component. Such a port **2430** would enable a user to set preferences through an external device or software application and would extend the capabilities of mobile station **2400** by providing for information or software downloads to mobile station **2400** other than through a wireless communication network. The alternate download path may for example be used to load an encryption key onto the device through a direct and thus reliable and trusted connection to thereby enable secure device communication. As will be appreciated by those skilled in the art, serial port **2430** can further be used to connect the mobile device to a computer to act as a modem.

[00324] Other communications subsystems **2440**, such as a short-range communications subsystem, is a further optional component which may provide for communication between mobile station **2400** and different systems or devices, which need not necessarily be similar devices. For example, the subsystem **2440** may include an infrared device and associated circuits and components or a Bluetooth™ communication module to provide for communication with similarly enabled systems and devices.

[00325] The embodiments described herein are examples of structures, systems or methods having elements corresponding to elements of the techniques of this application.

This written description may enable those skilled in the art to make and use embodiments having alternative elements that likewise correspond to the elements of the techniques of

this application. The intended scope of the techniques of this application thus includes other structures, systems or methods that do not differ from the techniques of this application as described herein, and further includes other structures, systems or methods with insubstantial differences from the techniques of this application as
5 described herein.

CLAIMS

1. A method for connecting a dynamic-content delivery infrastructure unaware generic element to a dynamic-content delivery infrastructure in a dynamic-content push-delivery system, the method comprising the steps of:
 - providing a dynamic-content delivery system mediator component between the generic element and the dynamic-content delivery infrastructure; and
 - the mediator component registering the generic element with the dynamic content delivery infrastructure.
2. The method of claim 1, wherein the generic element is a dynamic-content push delivery system unaware generic content provider.
3. The method of claim 1 or claim 2, wherein the providing step comprises provisioning a service provider from an external repository.
4. The method of claim 2, wherein the providing step comprises the steps of:
 - locating a generic content provider mediator component on a service provider dynamic-content delivery system component; and
 - obtaining, from the content provider mediator component, metadata for the generic content provider.
5. The method of claim 4, wherein the obtaining step comprises getting metadata from a central repository or a content provider specific repository.
6. The method of claim 1, wherein the generic element is a dynamic-content push-delivery system unaware generic application.
7. The method of claim 6, wherein the providing step comprises provisioning a mobile device from an external repository.
8. The method of claim 3 or claim 7, wherein the external repository is a generic repository or a content provider specific repository.
9. The method of claim 7 or claim 8, wherein each mediator component is specific to the dynamic-content delivery infrastructure.

10. The method of claim 6, wherein the providing step comprises the steps of:
locating a generic application mediator component on a mobile device; and
obtaining, using the application mediator component, metadata for the generic
5 application.
11. The method of claim 6, wherein the obtaining step comprises getting metadata
from a central repository or an application specific repository.
- 10 12. The method of any of claims 1 to 11, wherein the registering step utilizes
metadata.
13. The method of any of claims 1 to 11, wherein the providing step is user driven.
- 15 14. The method of any of claims 1 to 11, wherein the providing step is automatic.
15. The method of any of claims 1 to 14 further comprising, prior to said providing
step, the step of:
loading an application mediator component to provide a dynamic content delivery
20 aware environment.
16. The method of any of claims 1 to 15, wherein the generic element is one of a
content provider and an application.
- 25 17. The method of claim 2, wherein the generic element is a dynamic-content push-
delivery system unaware generic application and said registering step explicitly binds said
content provider to said application.
18. The method of claim 2, wherein the generic element is a dynamic-content push-
30 delivery system unaware generic application and said registering step implicitly binds said
content provider to said application.
19. The method of claim 18, wherein said implicit binding compares a content type
token provided by said application with a content type token provided by said content
35 provider.

20. The method of claim 19, wherein said comparing step counts a number of overlapping segments between the content type token provided by said application and the content type token provided by said content provider.
- 5 21. The method of claim 20, wherein said registering step utilizes said count from said comparing step to determine whether to bind an application with a content provider.
22. The method of any of claims 1 to 21, wherein said method further comprises, prior to said registering step:
- 10 adding delivery handling information for the generic element; and
converting information to a predetermined format.
- 23 The method of claim 1, further comprising extracting at the mediator component metadata for the generic element.
- 15 24 A dynamic-content delivery system mediator component for connecting a dynamic-content delivery infrastructure unaware generic element to a dynamic-content delivery infrastructure in a dynamic-content push-delivery system, the mediator component comprising:
- 20 registering means to provide registration information for the generic element to the dynamic-content delivery infrastructure.
- 25 25. The mediator component of claim 24, further comprising extracting means to extract metadata.
26. The mediator component of claim 24 or claim 25, wherein the generic element is a dynamic-content push-delivery system unaware generic content provider.
27. The mediator component of any one of claims 24 to 26, wherein a content
30 provider mediator is provisioned onto a service provider from an external repository.
28. The mediator component of claim 24, wherein the generic element is a dynamic-content push-delivery system unaware generic application.
- 35 29. The mediator component of claim 28, wherein an application mediator is provisioned onto a mobile device from an external repository.

30. The mediator component of claim 26 or claim 29 further comprising:
communication means adapted to obtain metadata for the generic content
provider.
- 5
31. The mediator component of any of claims 24 to 30, wherein the registering means
is adapted to utilize metadata.
32. The mediator component of any of claims 24 to 31, wherein the generic element is
10 one of a content provider and an application.
33. The mediator component of claim 26, wherein the generic element is a dynamic-
content push-delivery system unaware generic application and said registering means is
adapted to explicitly bind said content provider to said application.
- 15
34. The mediator component of claim 26, wherein the generic element is a dynamic-
content push-delivery system unaware generic application and said registering means is
adapted to implicitly bind said content provider to said application.
- 20
35. The mediator component of claim 34, wherein said registering means is adapted
to compare schema in a content type token provided by said application with schema in a
content type token provided by said content provider.
36. The mediator component of claim 35, wherein said registering means is adapted
25 to count the number of overlapping schema between the content type token provided by
said application and the content type token provided by said content provider.
37. The mediator component of claim 36, wherein said registering means is adapted
to utilize said count to determine whether to bind an application with a content provider.
- 30
38. The mediator component of any of claims 24 to 37 further comprising:
processing means adapted to add delivery handling information for the generic
element; and
converting means adapted to convert information to a predetermined format.
- 35

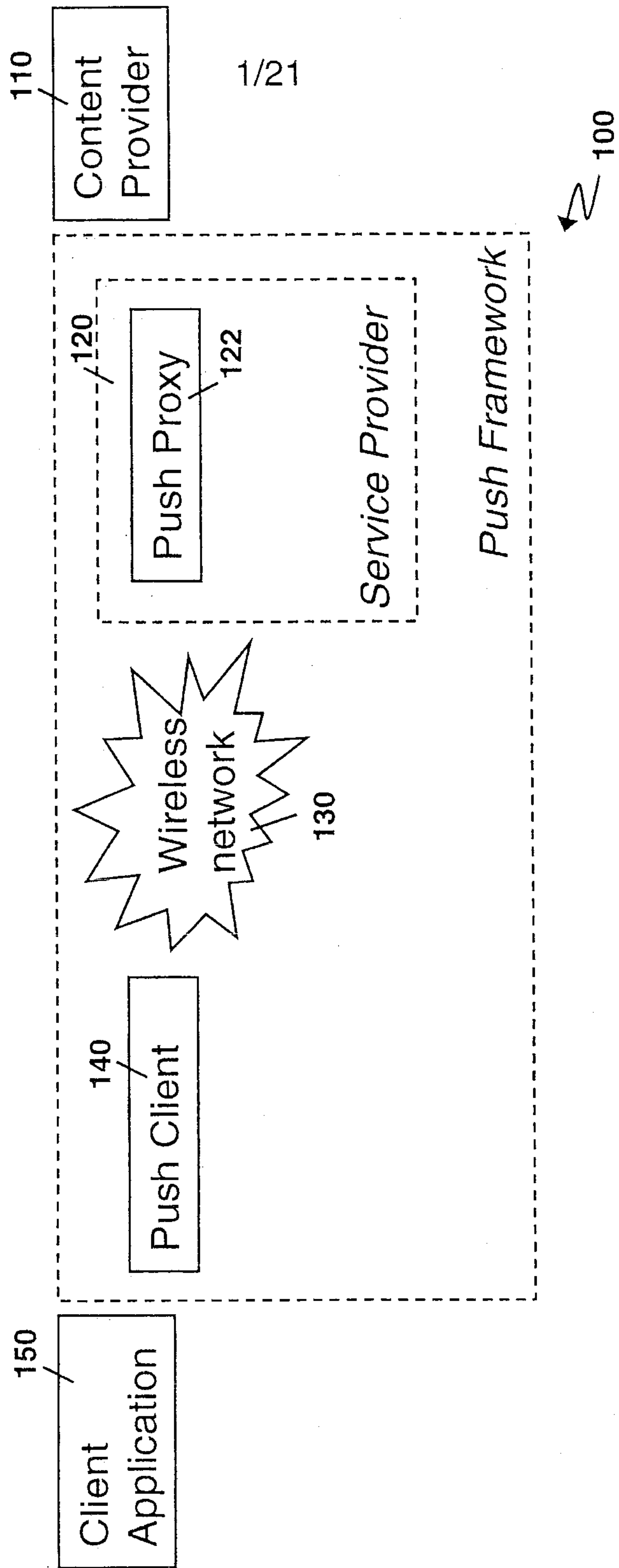


Fig. 1

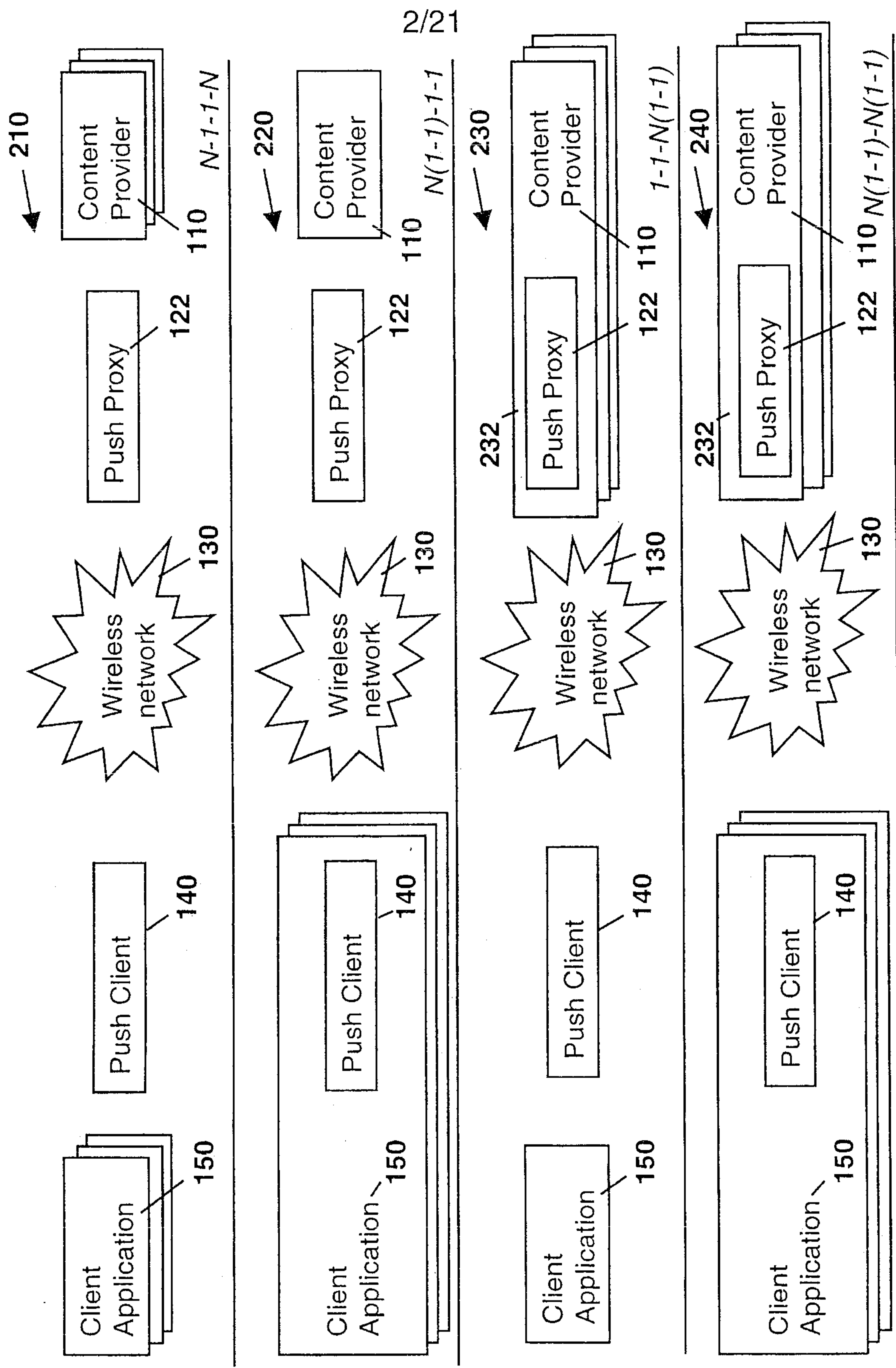


Fig. 2

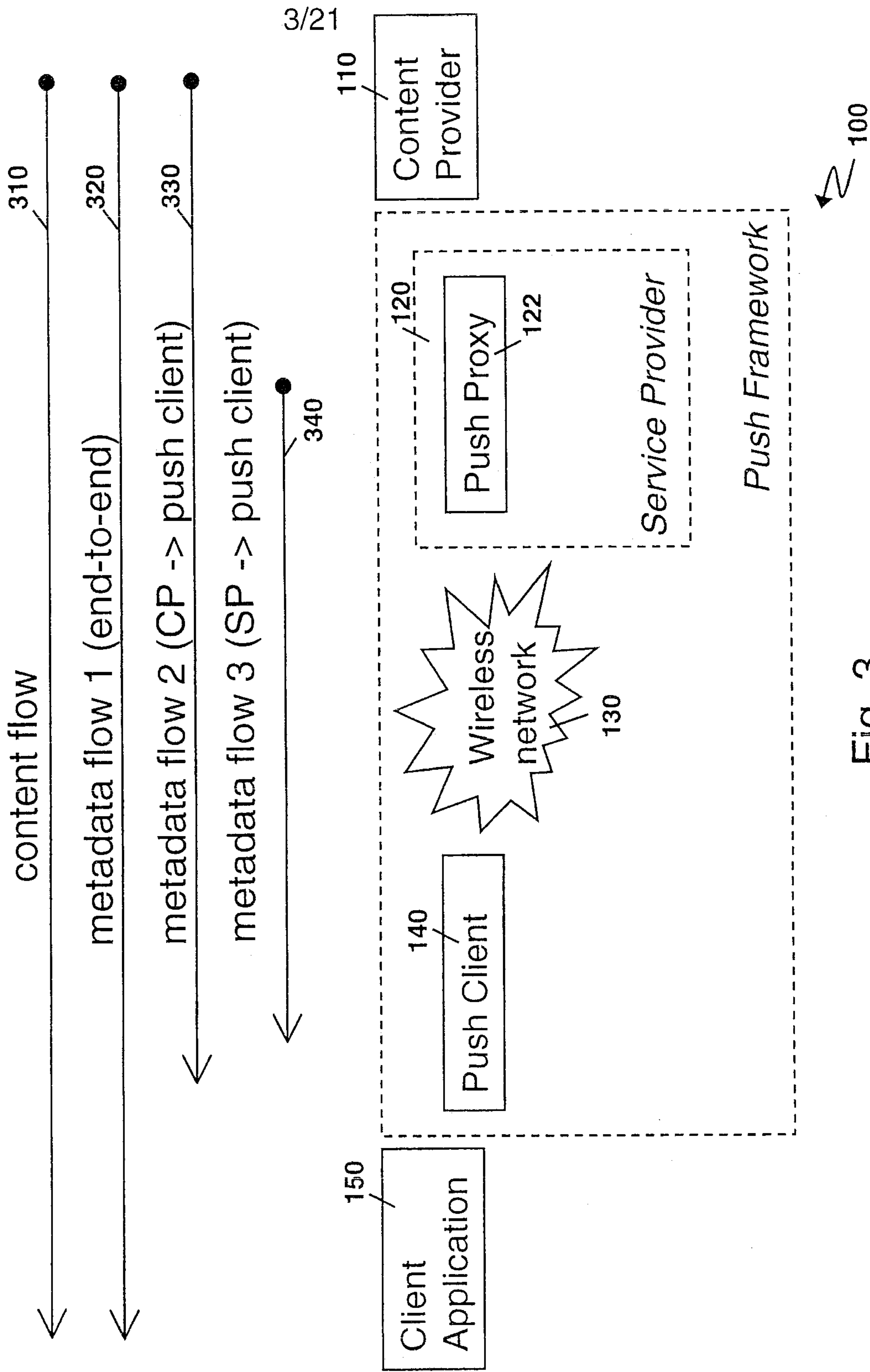


Fig. 3

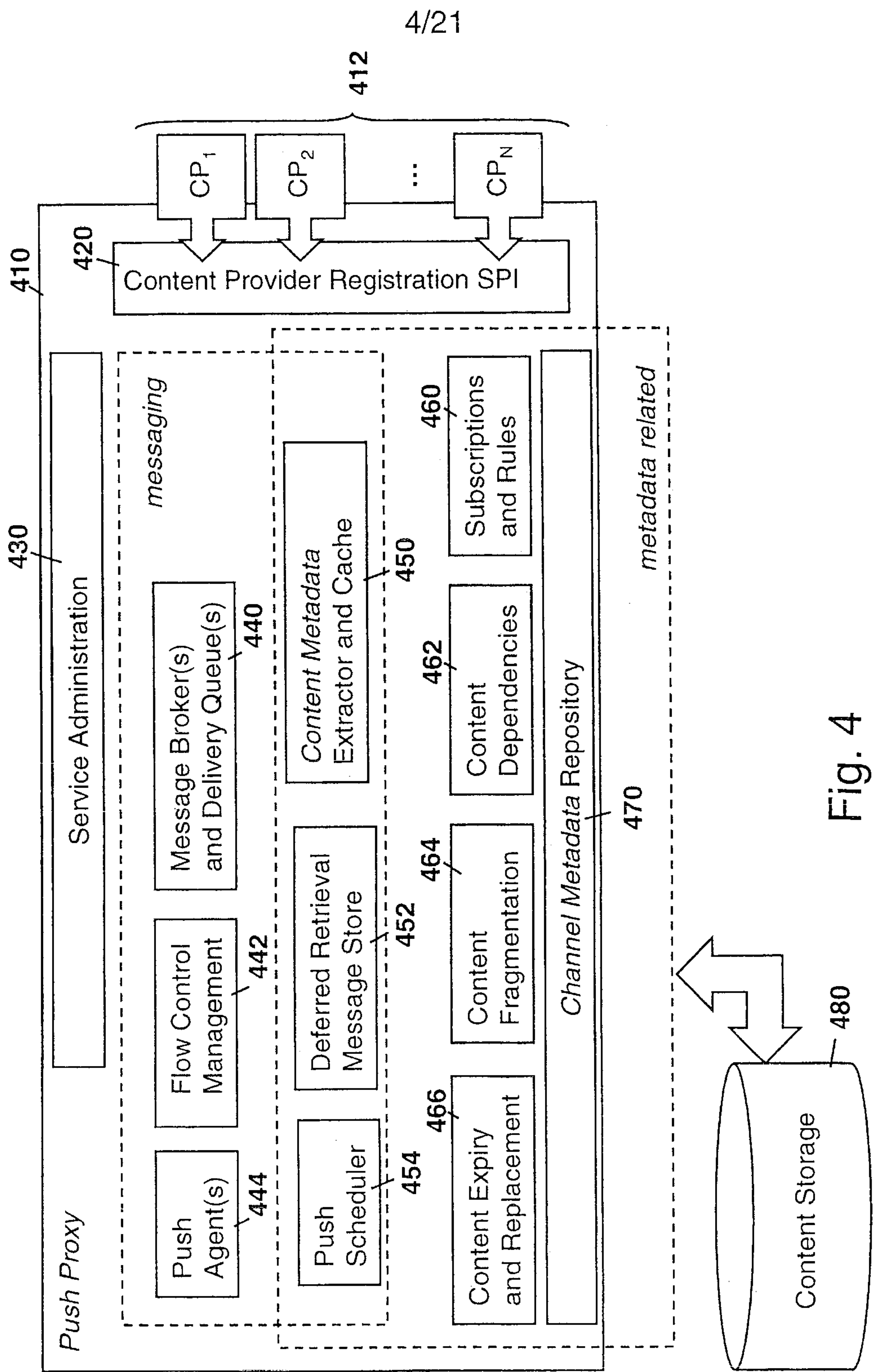


Fig. 4

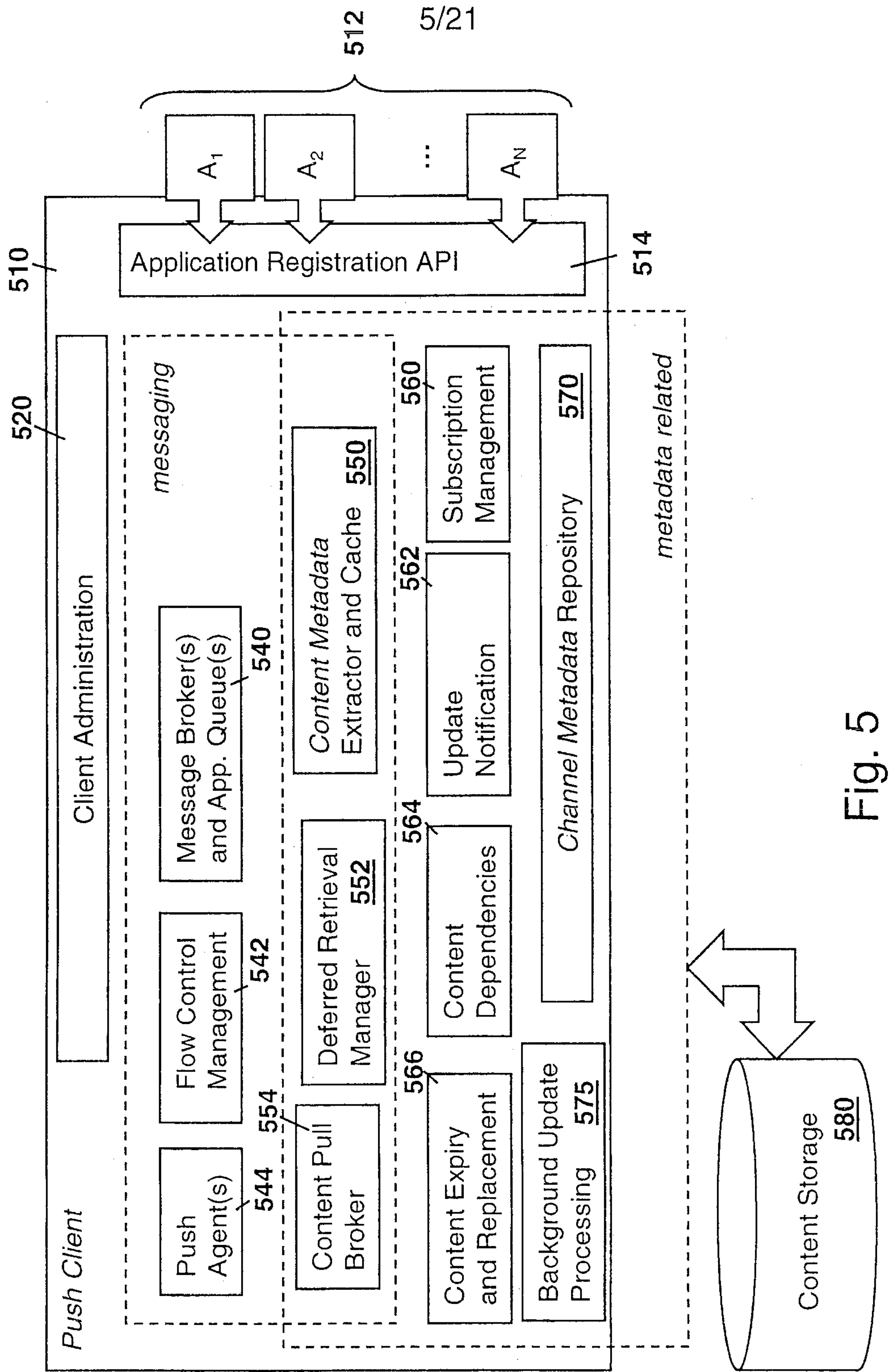


Fig. 5

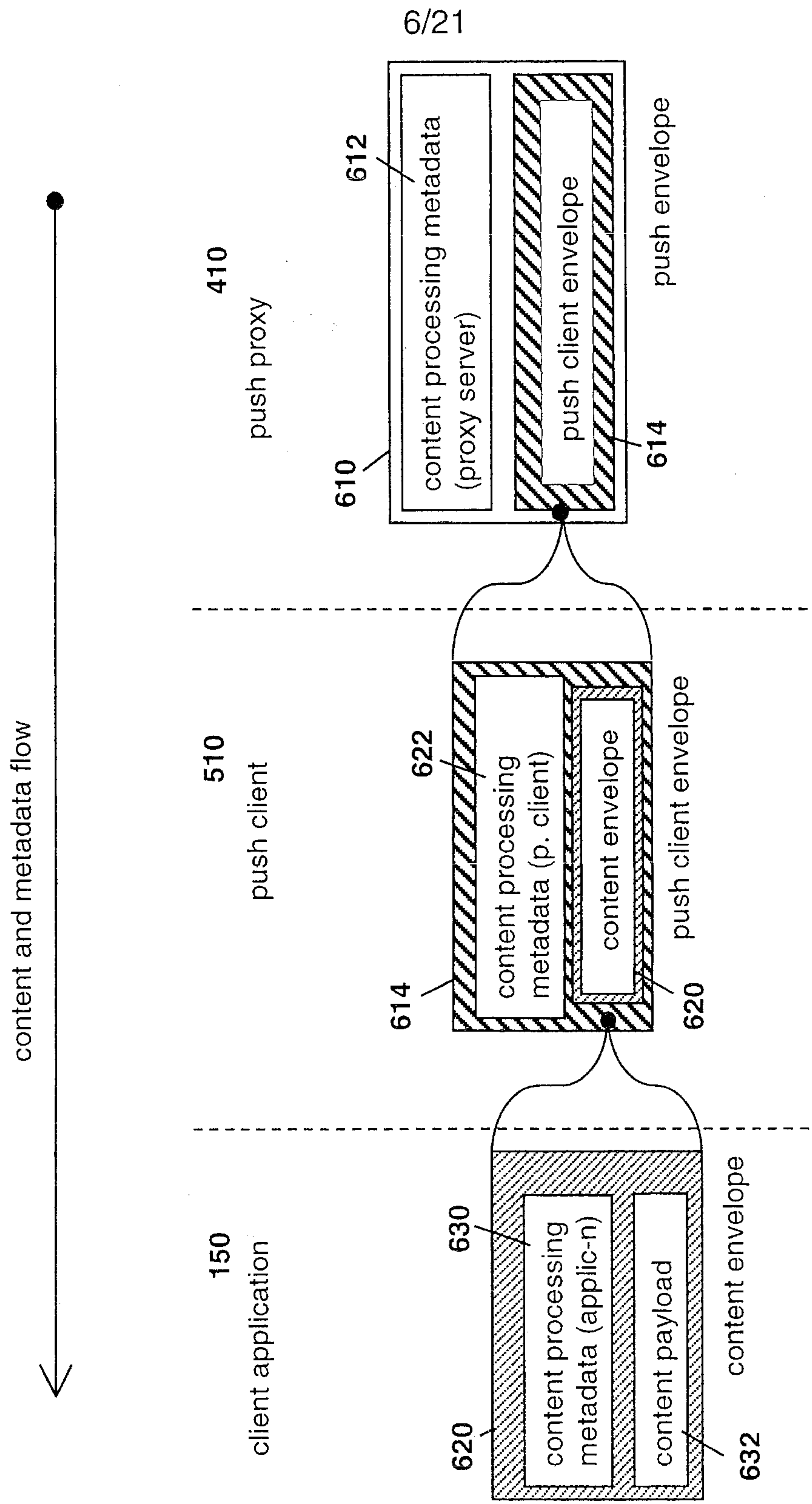


Fig. 6

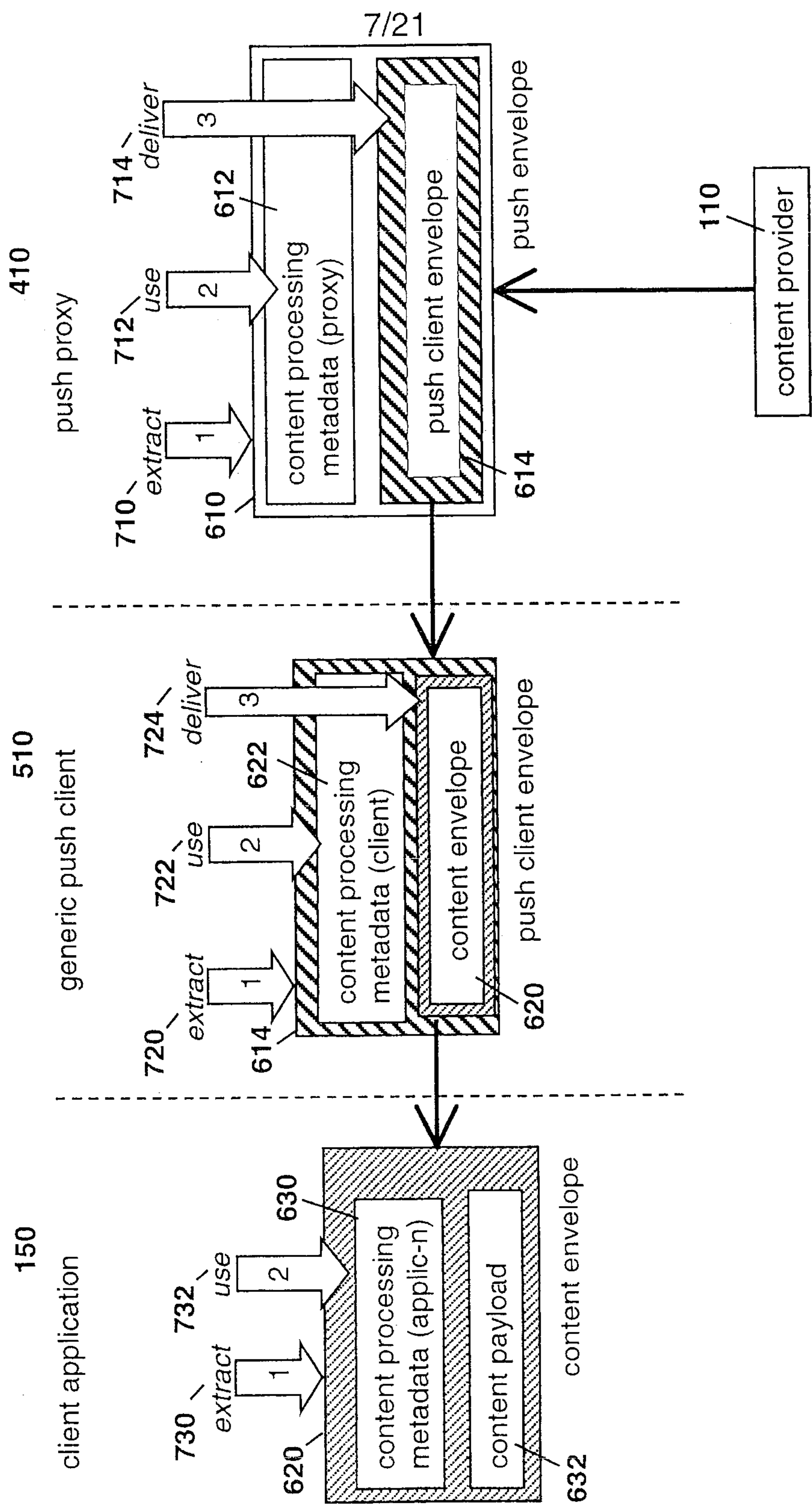


Fig. 7

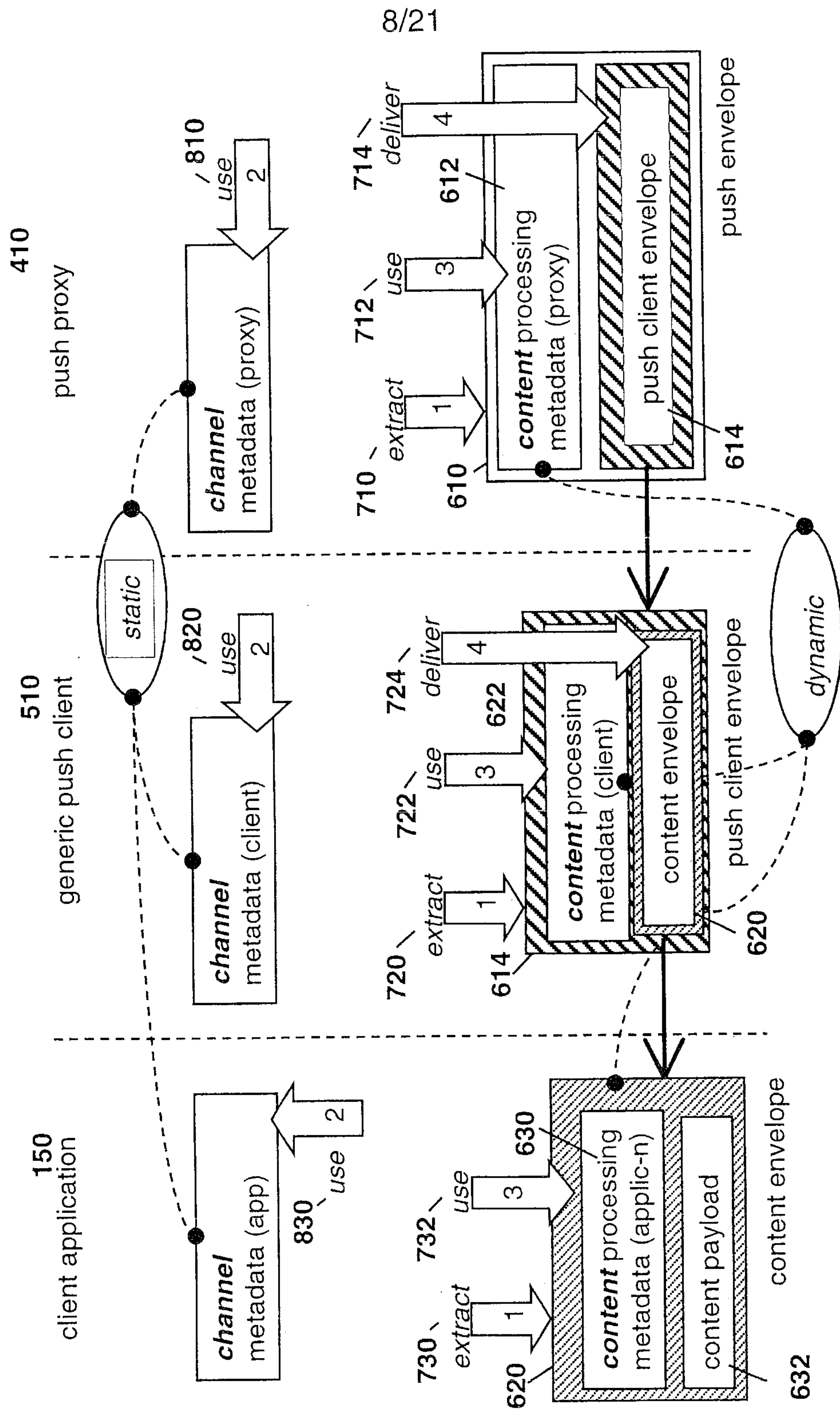


Fig. 8

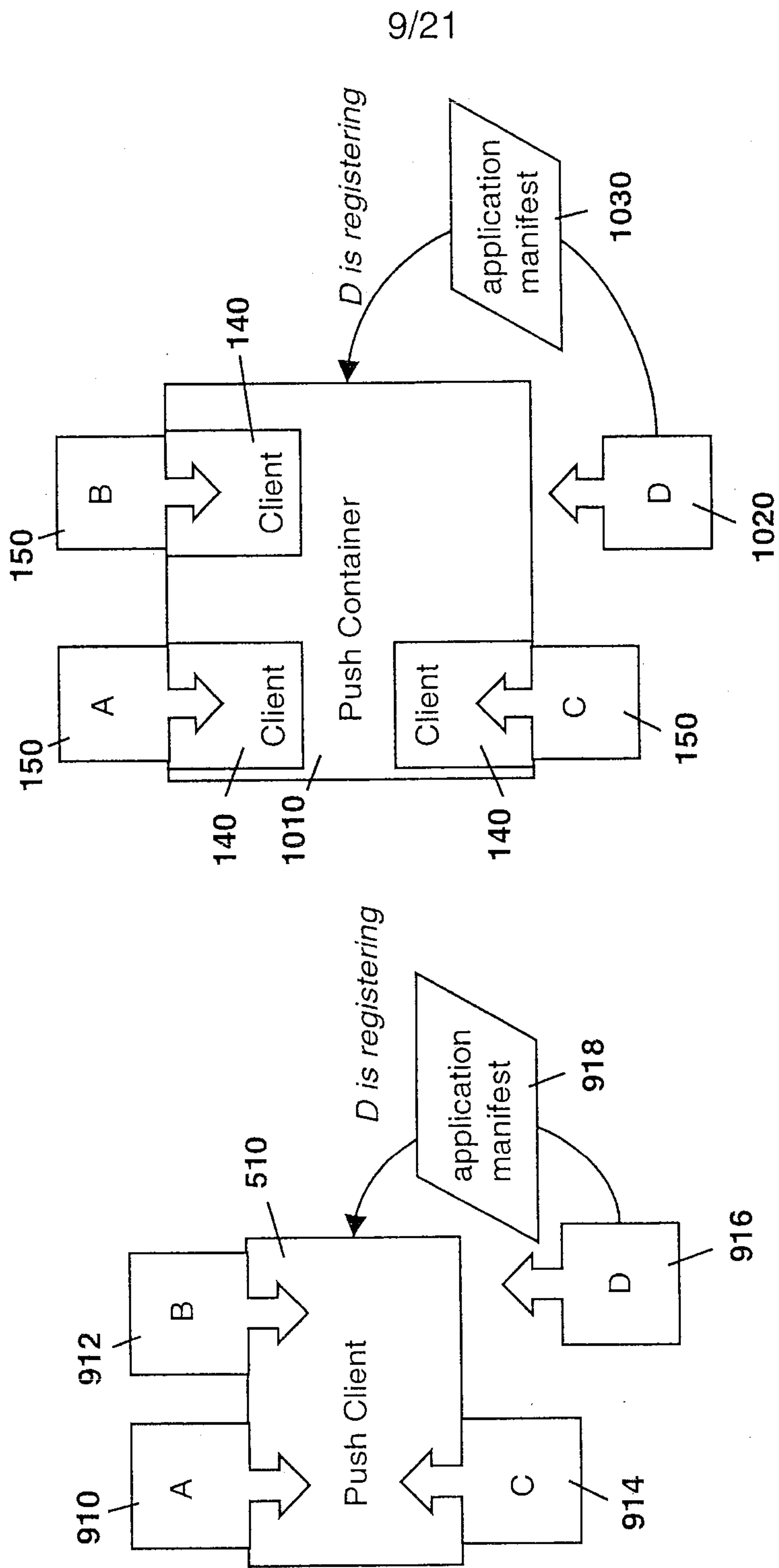
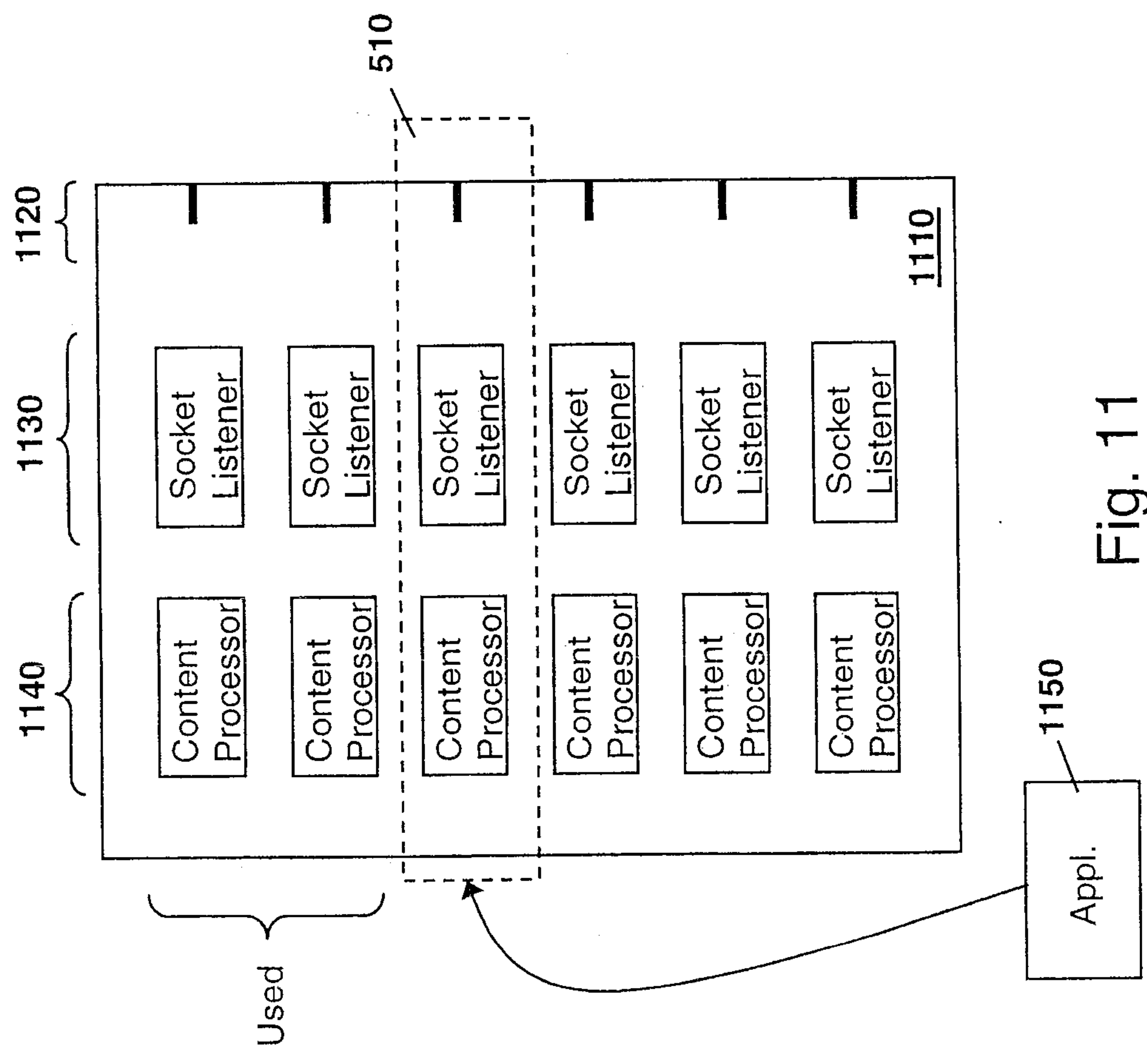


Fig. 10

Fig. 9

10/21



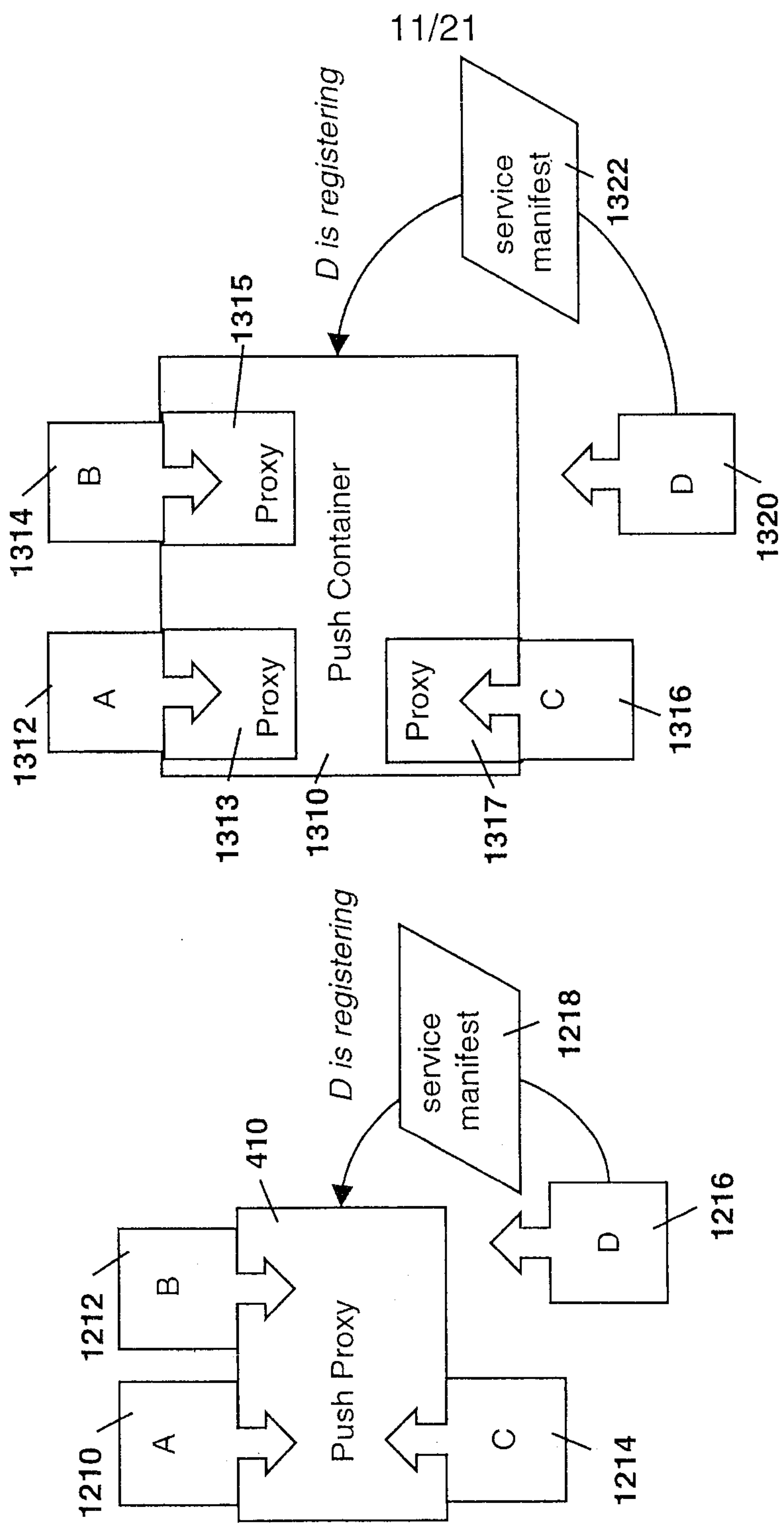


Fig. 13

Fig. 12

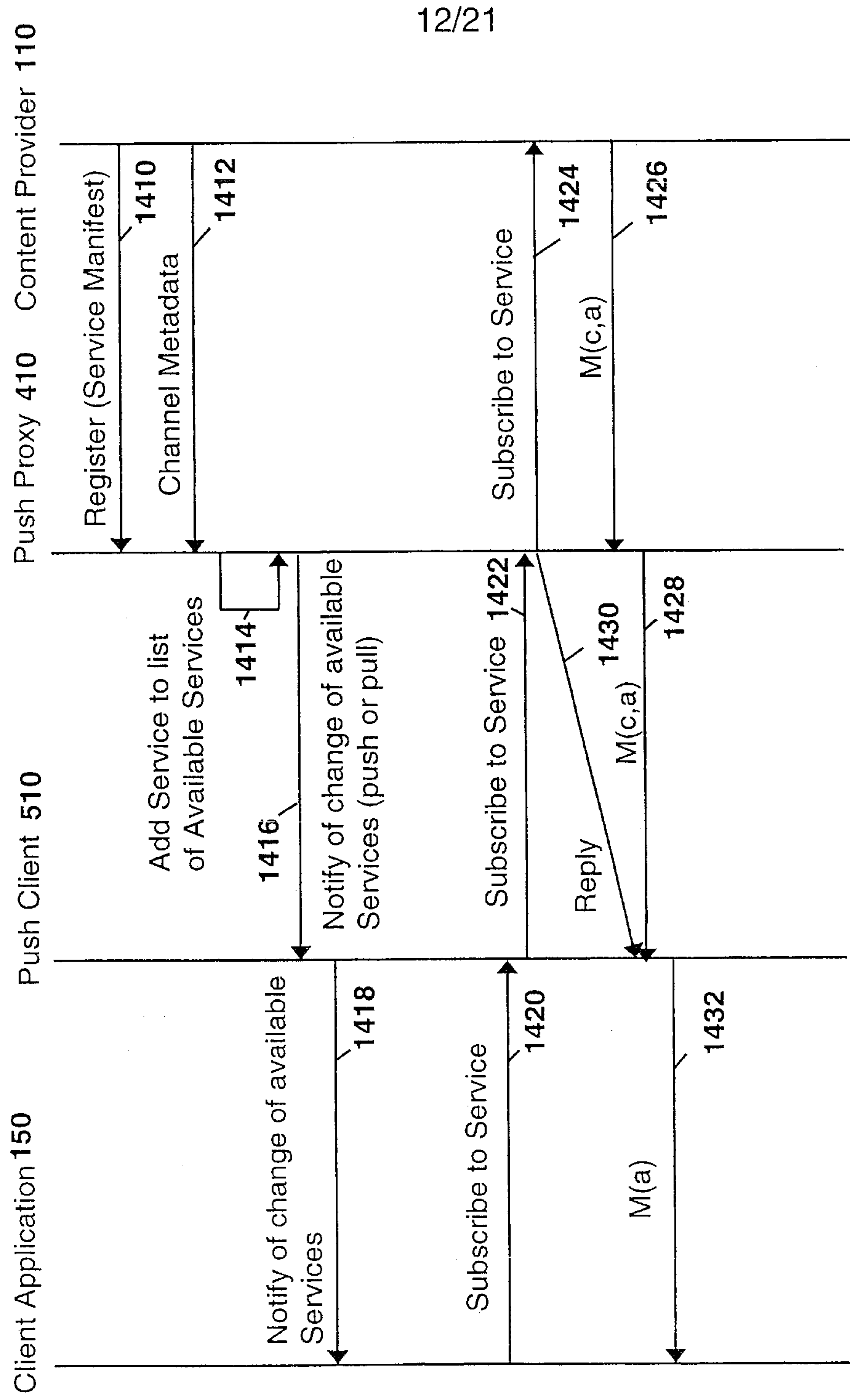


Fig. 14

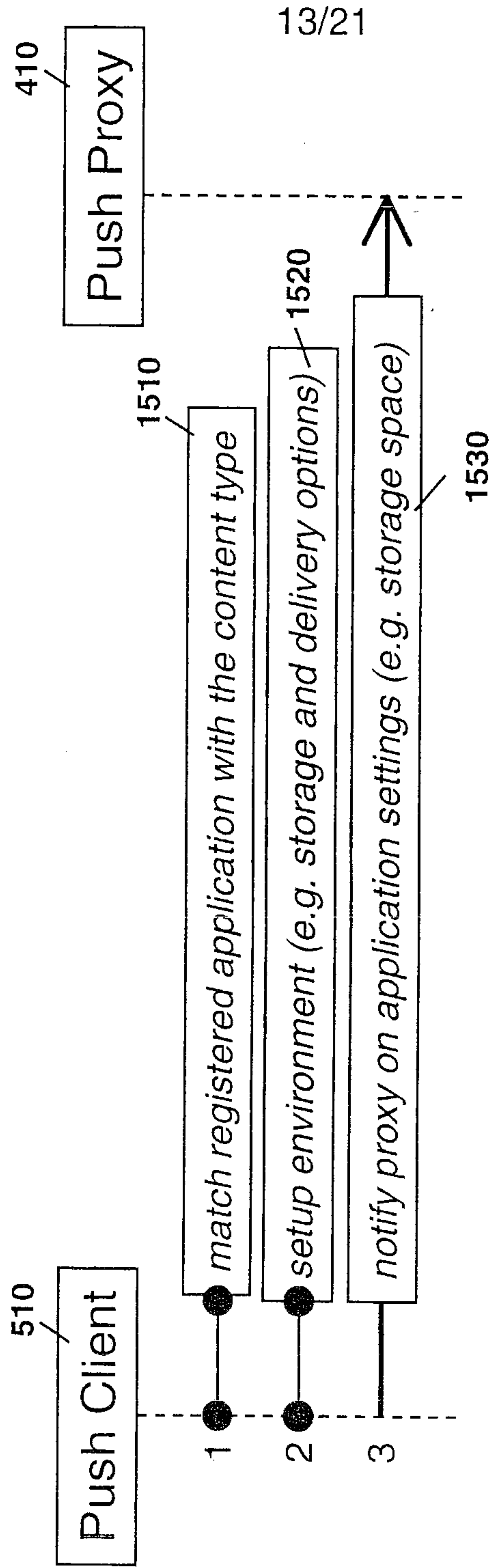


Fig. 15

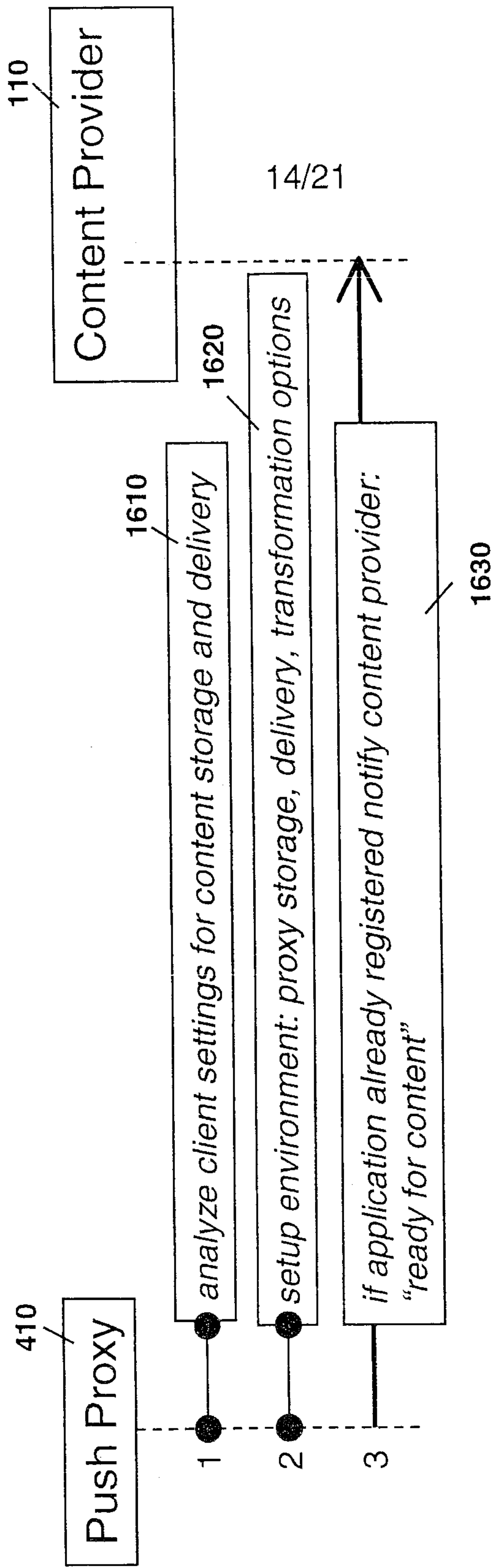


Fig. 16

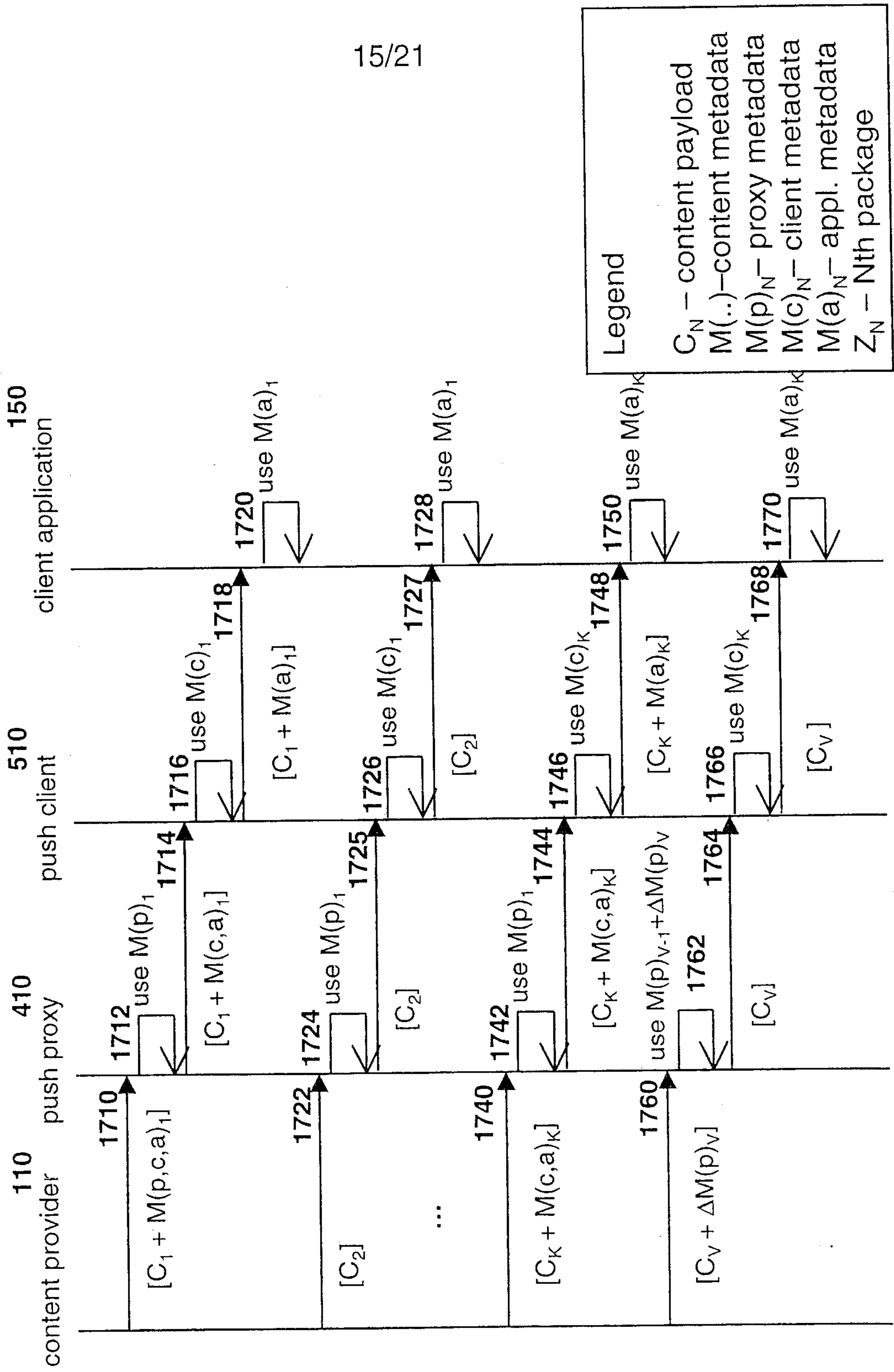


Fig. 17

16/21

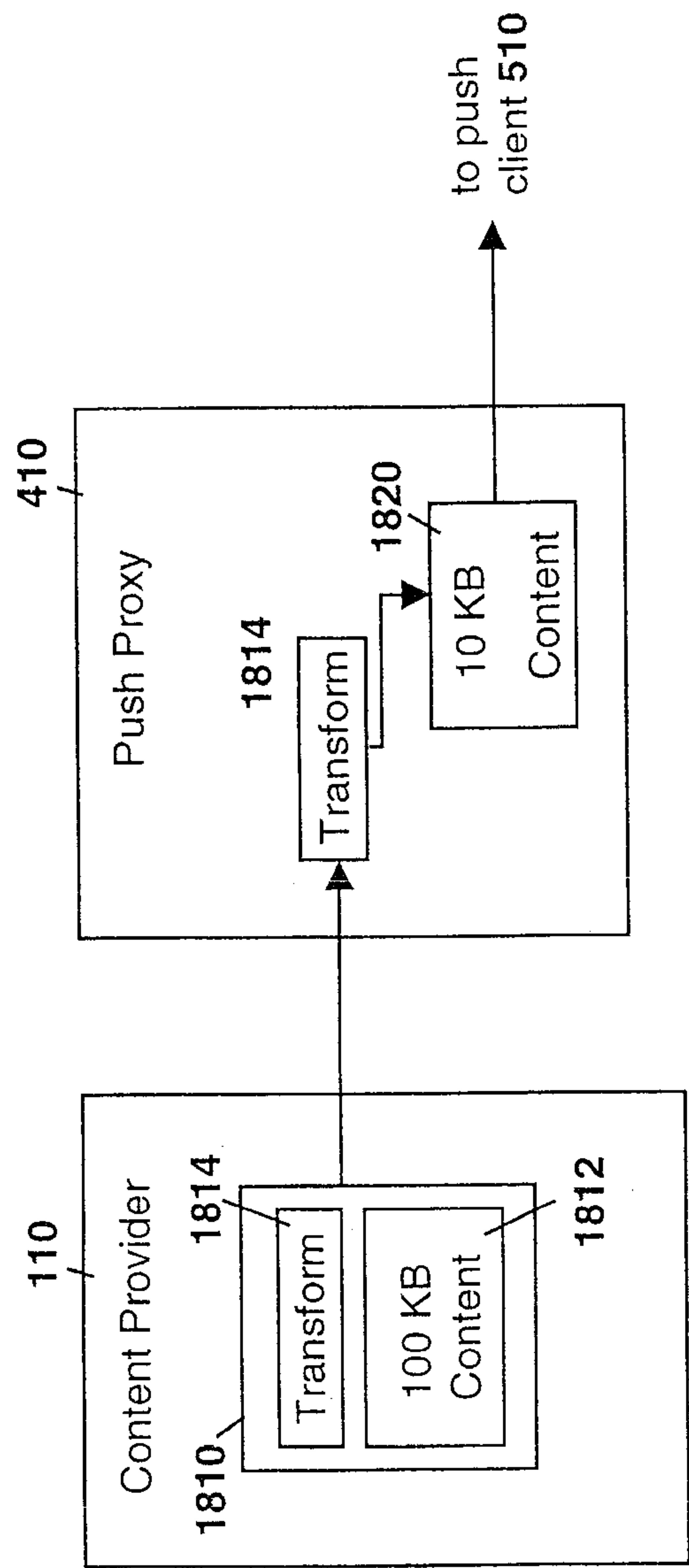


Fig. 18

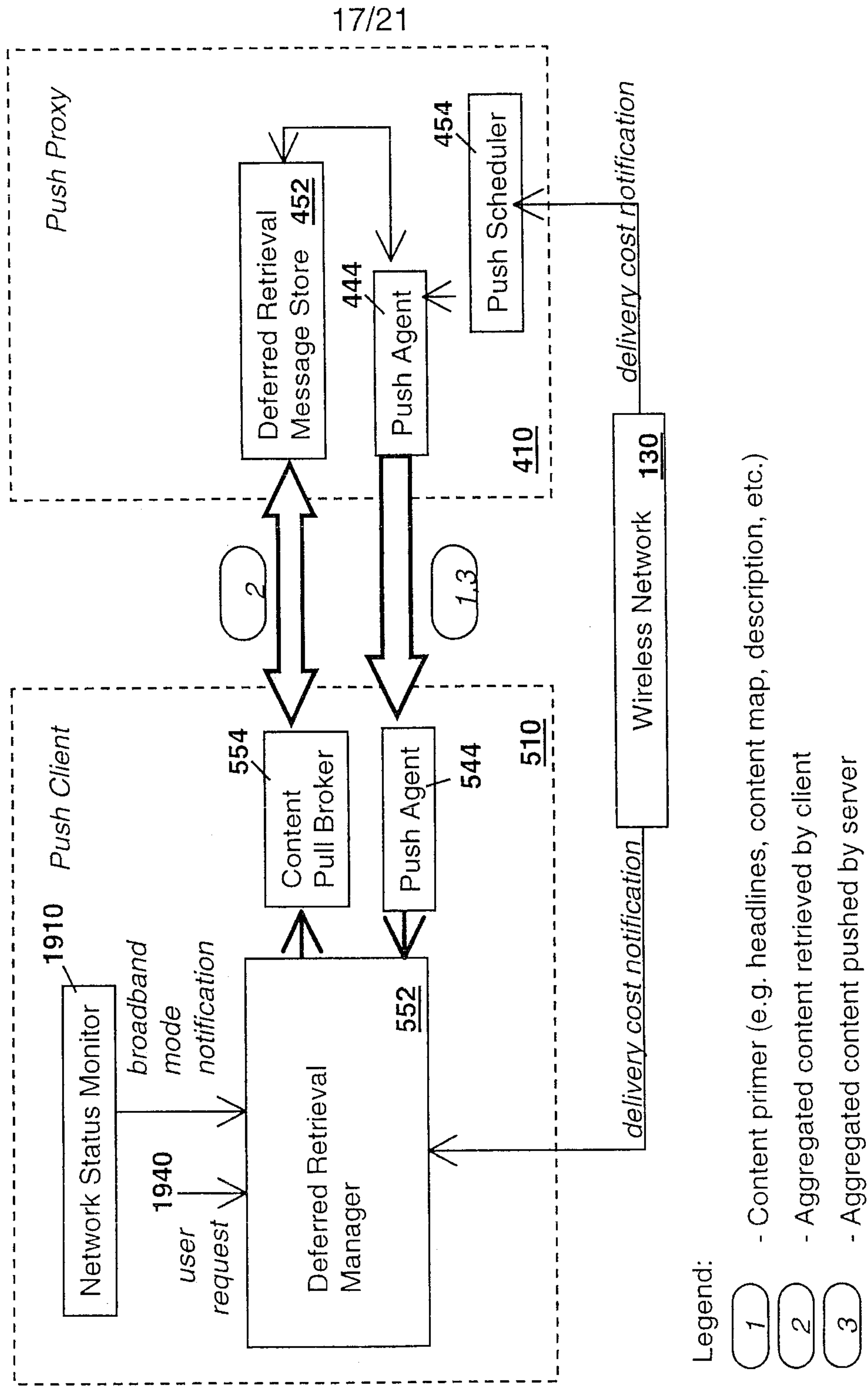


Fig. 19

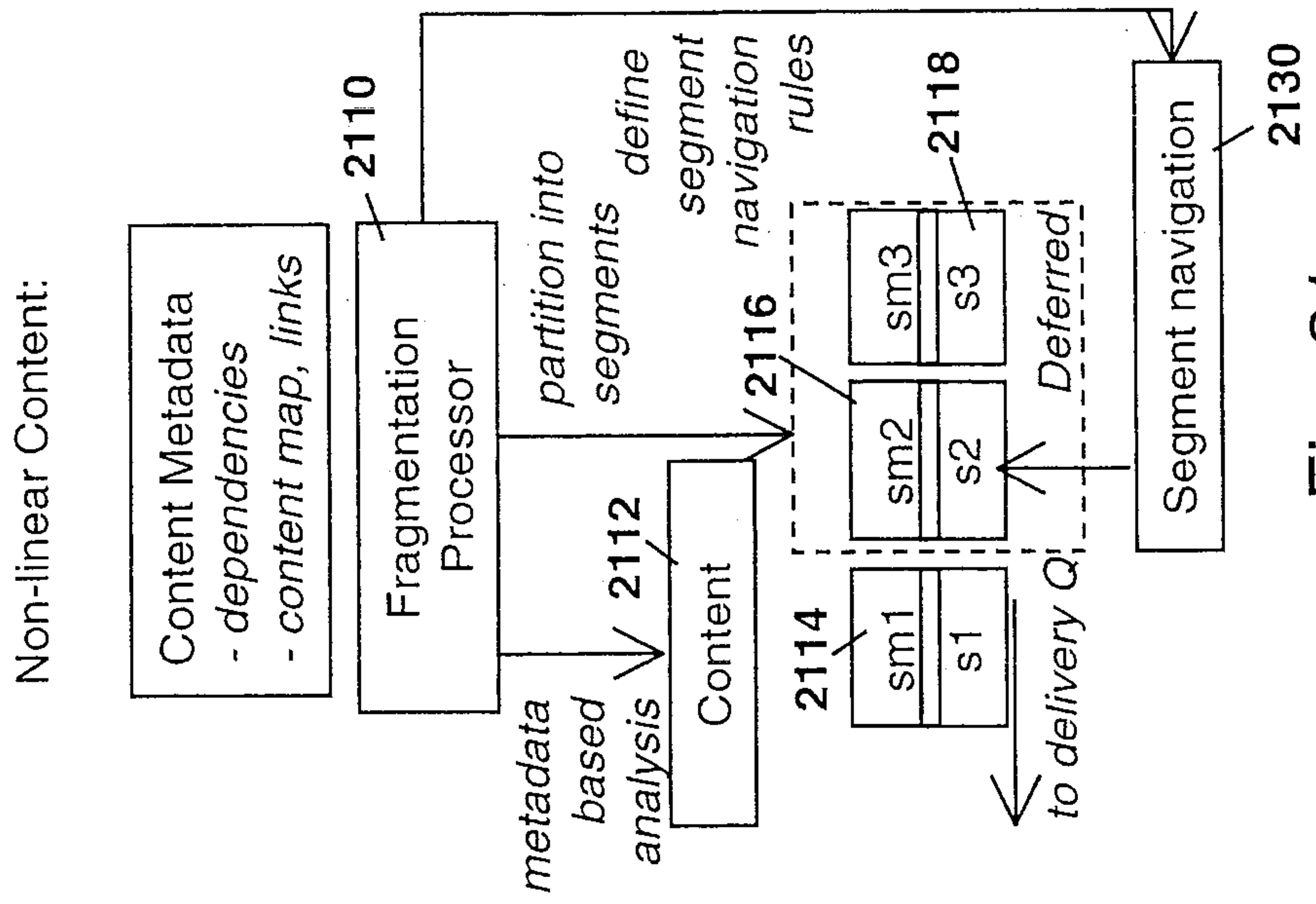


Fig. 20

- Legend:
- sX - segment X
 - smX - metadata (dependencies, map, nav. rules) for segment X

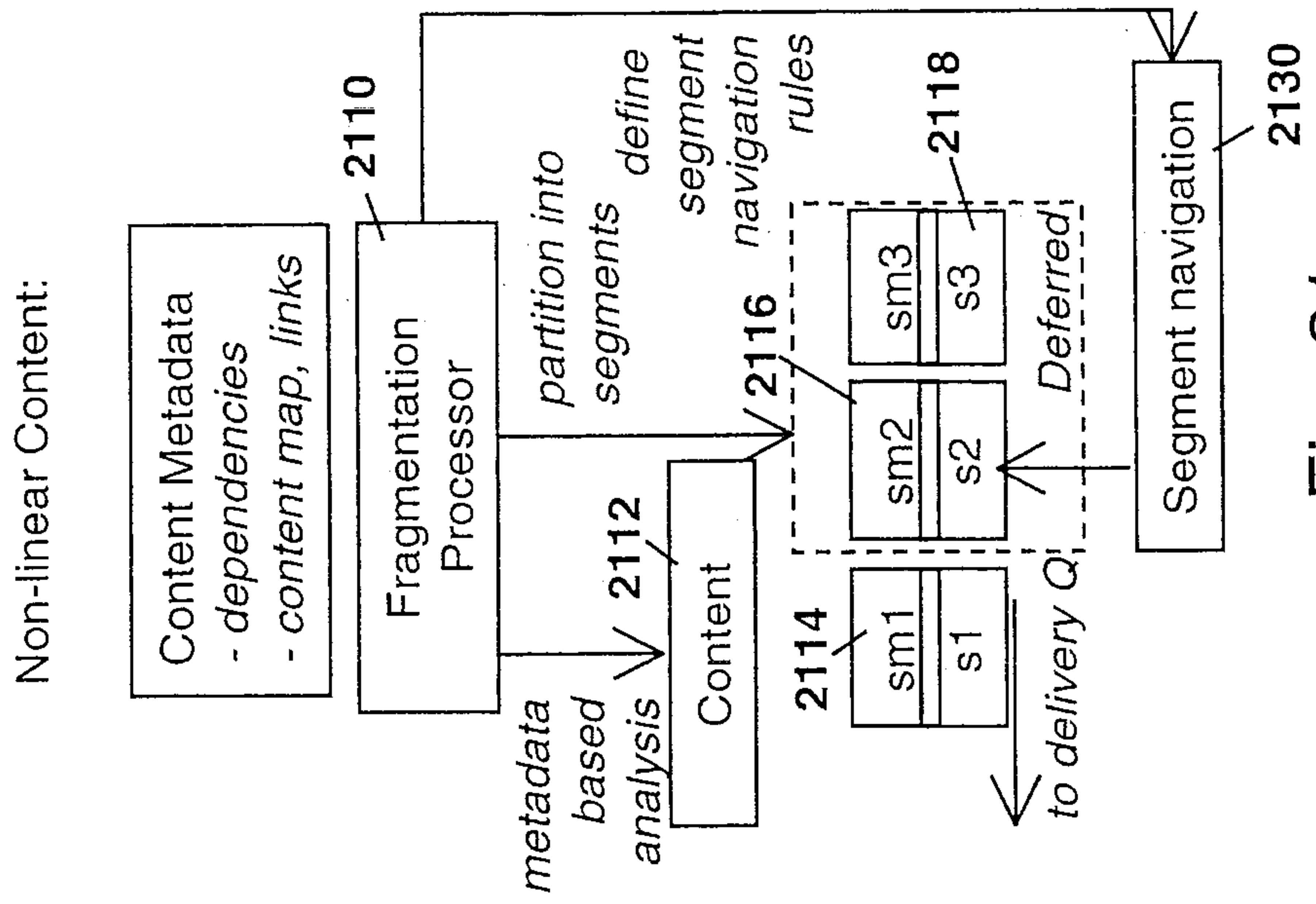


Fig. 21

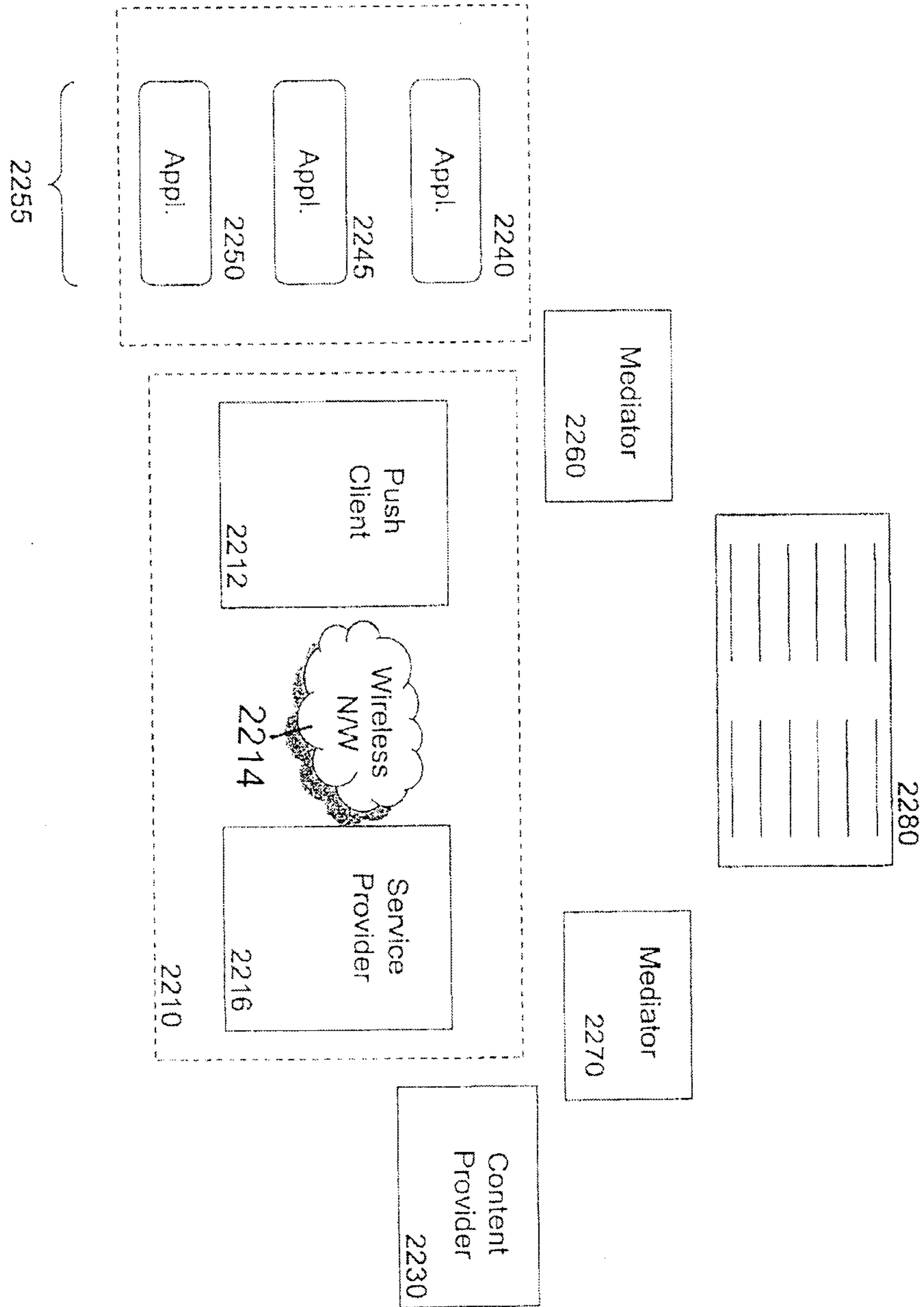


Fig. 22

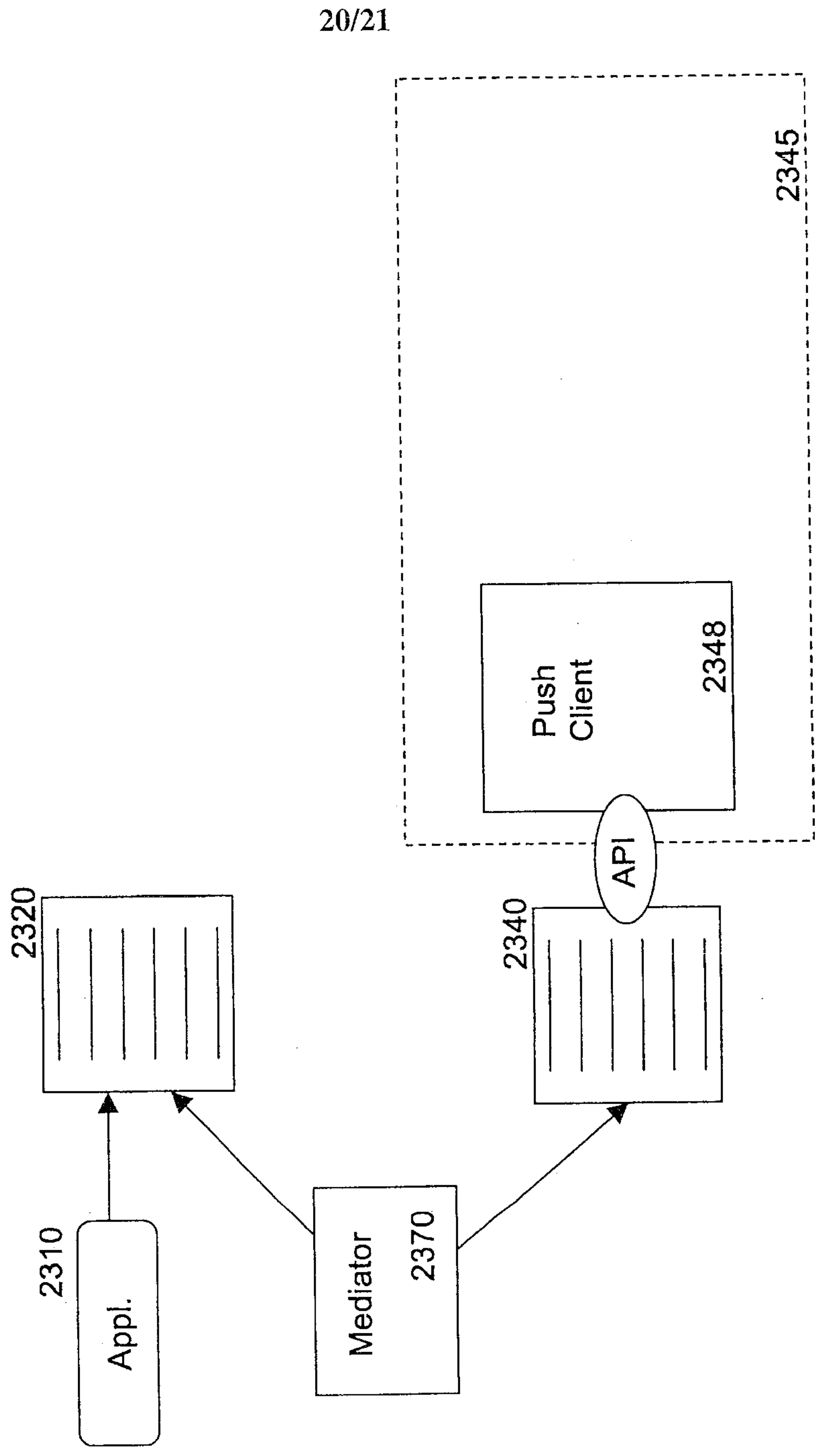


Fig. 23

21/21

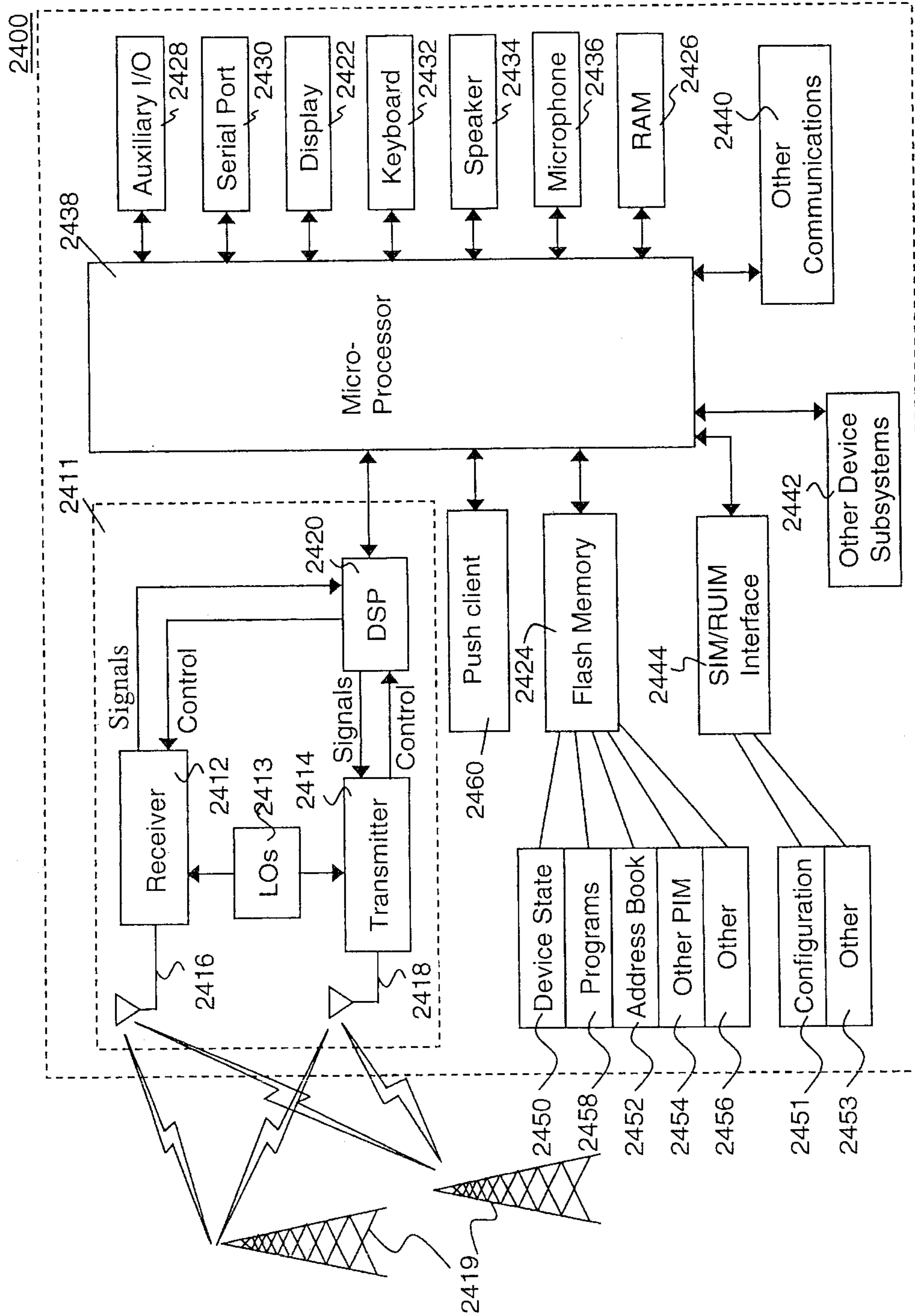


FIG. 24

