The Browserver*

Miguel Raposo
53952
Instituto Superior Técnico, Technical University of Lisbon
Av. Prof. Cavaco Silva, Porto Salvo, Portugal
miguel.raposo@ist.utl.pt

ABSTRACT

This paper presents the Browserver as both a human interface for web services and a generic universal platform for individually consuming and providing services. The empowerment of individuals and SMEs with a low cost workflow support platform that does not require expensive infrastructures, while ensuring information privacy and minimizing third-party dependency is the aim of the Browserver. The focus lies on shifting from client-server to a peer-to-peer (P2P) paradigm, enabling direct interactions between entities and eliminating the need for applicational intermediaries. Business scenarios are presented and a comparison is made between the use of the Browserver facing traditional solutions for inter-organizational and business to client interactions.

Keywords

Electronic Services, Human Services, User Interface, Browser, Server, Peer-to-Peer, Workflow, Collaboration

1. INTRODUCTION

Although there are several providers that offer free storage and collaborative applications that can be used as a basic business framework for small enterprises, there are major concerns and problems regarding:

- Privacy: information flows through central applications and servers, being co-owned by the entities that own them and is often available to more entities than those who are their actual proprietaries.
- Reliability: how reliable are the servers and Web applications we use? Can we trust they're going to be available when we need them?
- Workflow interactivity: a browser is only a client, therefore the user cannot be directly reached by the server. The server can not take the first step, and is dependent of the user's will to take initiative on starting an interaction, that way e-mail messages are still a common way of drawing the user's attention.
- Tool mismatch and lack of interoperability: simple collaboration tools are not integrated and were not made for business workflow support.

This paper presents the Browserver, which includes a browser and a server, both widely available, as a natural evolution for user interactions in the ever-growing Internet of Services. The basic objective is to allow individuals and small enterprises (from the larger down to the level of one person and one laptop) to interact directly, in a P2P fashion, without the need of exchanging possibly sensible information via central storage or server-based collaborative applications or even e-mail providers (Fig. 1). Each user becomes an active internet entity, able not only of invoking services but also of providing them (in particular, collaborative and workflow services) to authorized partners. The main motivations that drive the Browserver are:

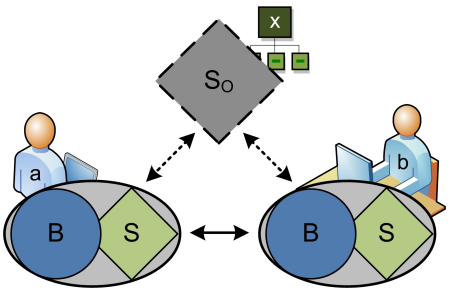

Figure 1: P2P Browserver. Central servers are not imperative for its interactions.

- Independence from third parties for service consuming and provisioning.
- Privacy of Web applications and collaborative tools.
- Direct and pair interactions between entities.
- Improve workflow interactivity for Web applications.
- Supply a platform for collaborative Web applications.

The Browserver constitutes a lightweight server based solution integrated with a browser and geared towards the end user, without the configuration and management effort of a fully-fledged server system, being installed as a complete application, much like a browser.

The rest of this paper shortly presents the background and the conceptualization, describes the architecture, implementation and a practical application of the Browserver as well as a conceptual evaluation over other solutions, the related work, draws the relevant conclusions and suggests some future Work on the platform.

2. BACKGROUND

2.1 World Wide Web

In the early Internet days, applications were made with specific client and server side components (Fig. 2) and specific protocols, with interactions limited by the existence of the specific client on each user's machine. Nowadays, the browser constitutes a generic, universal client component capable of accessing all of the ever-growing Web applications (Fig. 3). Web users are seen as information generators, not merely as consumers. Although services already constitute the main paradigm at enterprise integration and the Inter-

*This article is an extended abstract of the *Dissertation for the Degree of Master in Information Systems and Computer Engineering* of the same author, October 2010

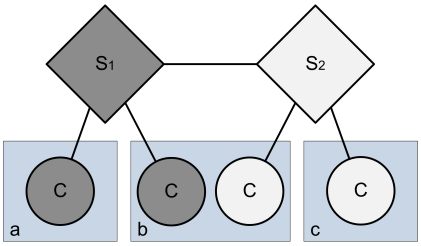

Figure 2: A Specific client for each specific application.

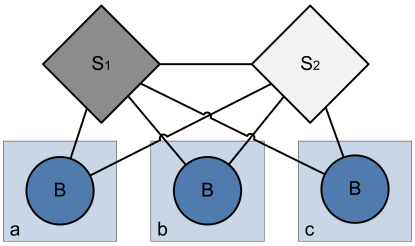

Figure 3: The Web browser, a universal client for Web applications.

net of Services [23] is already a discussion subject, the Web is still centered around content and not on services, with the client-server paradigm limiting the interaction patterns with humans by requiring these to initiate the interaction by navigating to some page through a URL. If a user is involved in some workflow, there is no direct way to interact with him through the browser so, the email is now the most used tool to contact and request someones services, having become a nightmare and not practical for many persons nowadays.

To reduce the client-server limitations, AJAX, polling and long-lived HTTP connections (Comet) [4] have been introduced to simulate server requests to the client, enabling more dynamic processes. Web Sockets [10, 11] are promising real bi-directional connections between the browser and the server, enabling better and faster communication between browser and server than AJAX. Nevertheless, the browser remains as a simple client, in the same paradigm, and business processes still depend on the user's will to initiate the interaction.

2.2 Email

The email remains as an indispensable tool to connect people, with its success mainly being due to its underlying fully asynchronous communication paradigm. To send information from person to person, there is no need for both to be available at the same time. Also, when sending an email, it is granted that it will eventually reach its recipient (if it exists) or a failure message will be sent back by the supporting email servers. That, allied to its universality (almost every person has and uses at least one email account) has turned it into one more piece to fill the gaps on interaction capabilities that the current Web model enables. Although its model is ideal for the way people operate on the Web, email overload [28] has been a problem that has not shown signs of slowing down [14] with its usage purposes far exceeding its basic original intent.

2.3 Services

A service is considered as a capacity exhibited by an entity (e.g. a user or system) which can be offered by him as provider, and invoked by other(s), as consumer(s). Each entity can be modeled by a set of services that represent their capacities, providing functionalities that enable the interaction of other entities with it. Composition of services is possible to create new and more complex services.

Web Services (WS) [17, 27] have been standardized by the World Wide Web Consortium (W3C), and consist of a set of technologies that follow the principles (standardized service contract, loose coupling, abstraction, reusability, autonomy, statelessness, discoverability and composability) defined for Service Oriented Architectures (SOAs). The Web Service Definition Language (WSDL) [3] which enables the service contract through service description while the Simple Object Access Protocol (SOAP) [19] is used for exchanging structured information independently of the transport protocol. The Universal Description, Discovery and Integration [1] system provides register and search mechanisms for services, enabling service discoverability.

RESTful Web services [22] refer to the design of systems based on a service paradigm (services in the form of resources) and strict utilization of REST [7] technologies. REST Web applications use generic interfaces and stateless interactions by transferring representations of resources based on their identifier (URI), instead of operating directly over those resources.

RESTful Web services provide higher performance and higher scalability but less expressiveness than the W3C Web services, which although suffer from the higher complexity and lower performance due to the numerous layers that compose it and the high verbosity of XML (its base technology).

2.4 Peer-to-Peer

The term peer-to-peer (P2P) is commonly and mistakenly associated with file transfer and sharing systems. Peer-to-peer means an architectural pattern for distributed systems were its main actors (the peers) are equally privileged, equipotent participants in the system. Instead of having to rely in central servers, peer-to-peer systems enable direct interactions between its participants. There can be more actors, or nodes, than the peers, who might provide indispensable functions for the whole or part of the system. The architecture was popularized by file sharing systems (like the extinct Napster or the popular and actual BitTorrent) but its application has no boundaries set for these.

3. THE BROWSERVER

The Browserver includes a browser (B) and a server (S), as represented in Fig. 4, hosted at a user's personal computer to ensure an active Web presence, with central servers reserved to host non-human based services and to support the logical peer-to-peer network of Web users, giving them the ability of being not only service consumers but also service providers. Interactions can be made directly between peers (a, b, c) equipped with a Browserver. Remote applicational servers ($S1$, $S2$) can also be accessed as usual but are not as crucial.

The browser acts as a user interface for locally hosted services that can be made available to the Web as well as to remote services that need to interact with the user. Each public service of the user can be directly consumed (called, requested) by another entity, turning every person into a service provider and the Web service centric and consequently user centric (as the services represent the user). The Browserver includes the following main characteristics:

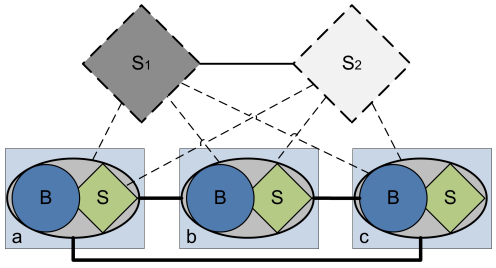

Figure 4: The Browserver, the union of a universal client and a universal server on P2P interactions.

- A generic application platform, suitable for almost any Web application (chat, e-mail, web sites, file transfer).
- A generic user interface for web services for executing services on user's demand, and for service's interaction with the user.
- Public and private services. The first can be published an searched in the web and the latter remain for private consumption or can be explicitly publicized by the user (by informing how it can be reached) to an entity to which he wants to provide the service.
- Publishing and searching for services. Distributed service directory(ies) enable the publication of services and searching for service providers.
- Relationship parity. Each Browserver is considered as a provider and consumer peer and is not restricted to one role of client or server.
- Fully asynchronous message communication which is granted by applicational (and informational) blind intermediaries (services Gateways).
- Direct P2P interactions. Push instead of pull. One can directly send requests to other's services (consume the services) without any specific applicational intermediary and on its own initiative.
- Unique service identification. Each service has its own unique universal identifier, that is related with its owner (provider).
- Information Privacy. Having none or only blind intermediaries, the information is kept private to their legitimate owners.
- Task management features including alerts for new and pending requests.
- User centric. The user (and not the applications) is the center of the platform and has control over his services.
- Suitable for Web standards.

4. ARCHITECTURE

The Browserver architecture includes both a local application, standing at each peer's computer, and the web on which it operates. The Browserver user can publish a service in the web so that it can be searched, discovered and consumed directly by another Browserver or a by a server from some organization. Figure 5 presents two simple cases where a user U_A publishes (*a1*, *x1*) descriptions of the services S_a and S_x to the web, where they are found and retrieved (*a2*, *x2*) by the Browserver user U_B and the organization O_c, through its server S_O. Interactions are P2P, with direct requests *a3* and *x3* followed by replies *a4* and *x4*, between services S_b and S_a and between services S_y and S_x.

4.1 The Application

The Browserver application couples a generic web browser and a server capable of executing web services. Possible architectural solutions include (a) developing a new web browser with an fully integrated server, (b) develop a new

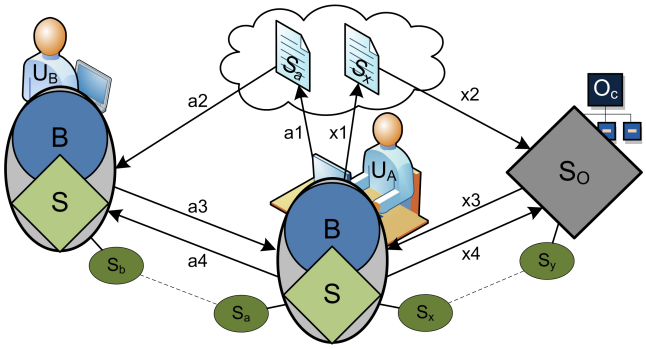

Figure 5: A user providing web services. Interactions with another Browserver and a server.

web browser connected to an application running on the server and (c) developing a server application to which any existing browser can connect. We present an architecture that aligns with option (c), which enables the use of any browser with any compliant server, while allowing normal web navigation. The Browserver application is divided into four main parts (Fig. 6):

- *Browser Management*: Manages the communication with the browser, updating the displayed interface. It receives UI requests from the Services Management and sends back related user replies;
- *User Interface Management*: Creates, destroys and manages all UIs requested by the Browser Management. Converts the internal representation of the UIs to a browser-displayable representation using XSLTs or specific converter objects;
- *Services Management*: Contains and manages the provided services and handles service's requests and replies;
- *Network Management*: Manages the network wherein the Browserver is. Enables service publishing operations on remote service directories and gateways as well as searching the directories for services.

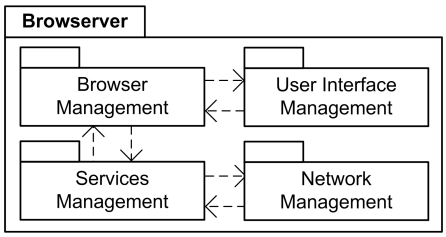

Figure 6: Main components of the Browserver application.

As shown in Fig. 7, the Browserver consists of a local server with a web application deployed that exchanges data *(a)* with a local browser. The browser acts as a User Interface (UI) for services deployed in the local server or in remote servers. Local and remote services can interact between themselves *(c, e)* or with the user *(b, d)* through the Browserver UI services.

4.1.1 Base Services

Base services provided by the Browserver, and managed by the *Services Management* component, include:

- *UIWebService*: A generic web service enabling the creation of UIs to interact with the user, with one operation (*requestUI(ui)*) that receives a UI definition that

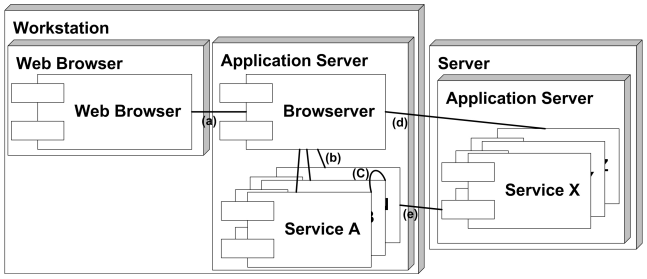

Figure 7: Architectural deployment view of the Browserver.

will be presented to the user. The execution of this operation has no immediate result. The user's reply is sent as a new message to the specified reply endpoint, complying with the fully asynchronous interaction pattern of the Browserver.

- *SystemWebService*: Provides system operations for user' services, such as *deploy(uri)*. Depending on the implementation, the *uri* parameter of the operations might be the USI (described in section 4.2) or the web service definition's (e.g. WSDL) URL.

The *UIWebService* offers a generic means of presenting interfaces to the user. Specific UI services can be developed using this generic service. A *requestAge()* operation of a UI service makes use of the *requestUI(ui)* with an UI definition to present a form with a simple text and field for the user to fill in and submit. The interactions that happen for UI requests from both a local service *X* and a remote service *Z* to the Browserver are shown in Fig. 8. Both requester services define a reply endpoint (*ReplyWS*) to receive the asynchronous response to the requests *(a1, b1)* made to the *UIWebService*. The *ServicesManager* receives *(a2, b2)* a *UIRequest* from the *UIWebService* and sends *(a3, b3)* the *UIData* that contains the UI definition to the *BrowserManager*. If there is a user response (a form submission) it is sent *(a4, b4)* to the *ServicesManager* who builds a *UIResponse* based on the UIRequest provided information and dispatches the message to the reply endpoint *(a5, b5)*.

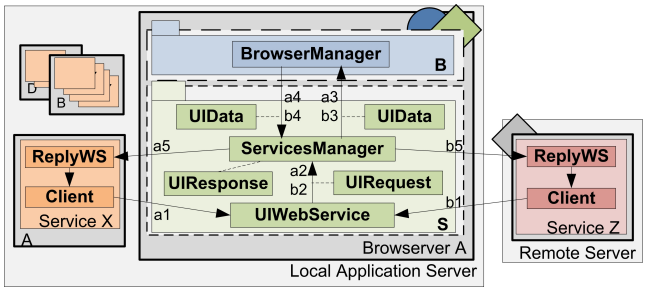

Figure 8: Interactions on UI request and reply.

4.1.2 Interface Generation and User Interaction

The generation of interfaces for services enables the interaction of the user with web services [25, 12] (with an HTML page with the necessary parameters from a service description (e.g. WSDL) being generated) or to support the interaction with the user under the initiative of an external entity, which can consume its *UIWebService* and send the UI definition (simple informative text or complex elements such as forms) to be presented to the user. A simple XML interface definition for a name and age request is presented in listing 1.

Listing 1: UI definition sample

```
<interface title="Information">
  <form name="informationForm">
    <par>Please fill and submit the following
      information:</par>
    <text>Your Name:</text>
    <input name="name" type="text"/>
    <text>Your Age:</text>
    <select name="age">
      <option value="< 20"/>
      <option value="20-40"/>
      <option value="< 40"/>
    </select>
  </form>
</interface>
```

The UI definition in each request is parsed by a *UnitBuilder* and transformed to an internal uniquely identified UI representation *Unit*. Each *Unit* is placed into a *Container* which holds all active UIs. The default container of the Browserver is a *Portal* which holds the UI *Unit*s as *Portlets* and builds an interface with a Desktop-like metaphor. The internal UI representation is transformed in a definition that is interpretable by the browser. The default *Portal* and *Portlet* conversion generates HTML, CSS and Javascript code.

To contextualize UI requests, and reuse or replace an existing *Unit*, each UI request has an associated *relatesTo* unique identifier (UID). In Fig. 9, the message *(2)* sent by service *X* to the Browser requests a UI with no *relatesTo* UID and as result, a new Unit is created. The response *(2)* is sent with an ID equal to the Unit to which it is related. The next request *(3)* relates itself with the previous reply, resulting in the reutilization of the Unit that changes its ID to a new one, which is again sent in the reply *(4)*. This way the same visual interface unit is used for consecutive interactions with the user, keeping him in the same execution context.

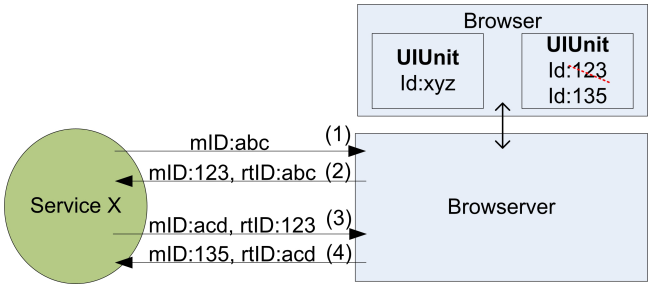

Figure 9: Interactions on UI request and reply.

4.1.3 Implementation

The Browserver has been developed as a Java web application using Java Servlets, Beans, Plain Old Java Objects and W3C SOAP-based Web Services. A Glassfish server and a standard browser have been used for the deployment of the application and display the UIs for the services, respectively. Glassfish supports the deployment of new applications without restarting the server, which allows adding new services as small web applications without restarting the Browserver. Firefox, Chrome, Internet Explorer and Opera have been tested and work with the developed solution.

To exchange data between the browser and the Browserver and to update the UI displayed to the user, the implementation uses AJAX techniques [4] to actively connect the Browserver to the browser, enabling nearly real time update of the UI on any event at the Browserver.

The Direct Web Remoting (DWR) [15] library has been used for reverse AJAX communication between the browser and

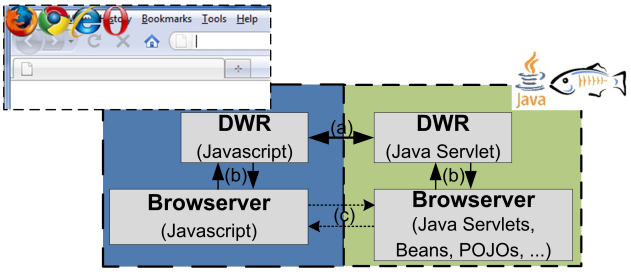

Figure 10: Connection between the browser and the server through DWR.

the Browserver application and JQuery [21] Javascript at the browser side as auxiliary to update the DOM structure of the visualized page and gather form data that is submitted through DWR. As shown in Fig. 10, The Browserver communicates *(c)* with the browser through DWR *(b)*, which consists of server side Java Servlet in communication with client side Javascript methods through reverse AJAX *(a)*.

The implemented *Container* of the Browserver is the *Portal* (Fig. 11) which holds *Portlets* (*a,b,c*) as the *ContainerUnit*s and builds an interface with a window system like metaphor, achieved through HTML, CSS and JQuery. Through the control panelit is possible to open the list of services *(b)* that are deployed in his Browserver and a list of all requests yet to be attended *(c)*. The orange Portlet *(a)* corresponds to the information request of listing 1 and as it was made by an external entity, it is represented with a different color. When a new request arrives, a new portlet is created and shown to the user in a blinking state. If the browser is closed it is opened, if it just not active (the portal does not have the focus of the user) an alert window is opened to draw user's attention.

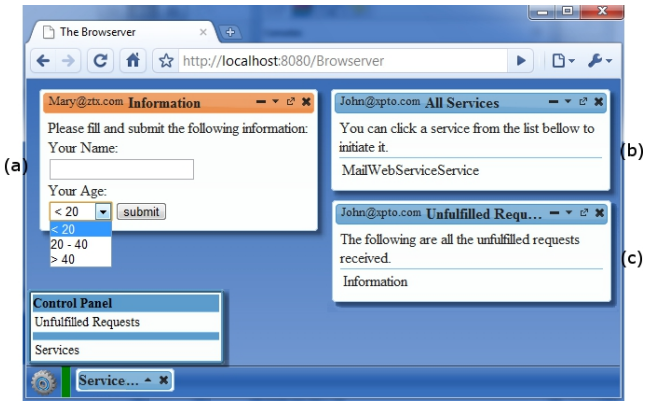

Figure 11: The Browserver portal.

4.2 The Web of Services

Users change their location (with DHCP) and their computers are not always on and connected. A central server (the Gateway) is used as an infrastructure to support a continuous Browserver minimum presence. For publicizing and searching services, we can use distributed web service directories [6], a distributed directory based on UDDI or a federated directory system for semantic web services based on OWL-S descriptions [24].

4.2.1 *Universal Service Identifier*

As a person is unique, so are the services he provides. Although two users might provide the same capabilities, the fact that they are not the same person makes the most basic distinction between both. A universal service identifier (USI) provides a means of univocal identification between services. The Browserver USI is a subset of the URI [2], with the syntax:

$$usi : [Provider][/Service][/Operation] \quad (1)$$
$$[Provider] = [name]@[subdomain].[domain] \quad (2)$$

The following are examples of Browserver USIs:

- usi:mike@ist.pt/
- usi:mike85@ist.pt/Accounting
- usi:john@chunk.us/MathService/squareOp

Each public service must have its own USI and be registered at the USI system (USIS), that offers *register*, *replace* and *unregister* operations through a web service. The USIS periodically pings every registered service, to detect faults and define its availability status. If an entity needs to be notified of the availability of a service, it can subscribe to the service availability notification provided by the USIS. If the location of the service description changes, it is the Browserver's responsibility to update it in the USIS.

The distributed USIS (DUSIS) system relates USIs to WSDLs locations, as exemplified in the following table from a node:

Table 1: DUSIS node table

Browserver	Service	WSDL Location
John@xpto.com	UIWebService	http://x.x.x.x:y/BrowserverGateway/res/95f05c48.wsdl
Mary@xpto.com	UIWebService	http://b.b.b.b:c/Browserver/UIWebServiceService?wsdl

If the USI is not present in the primary node's tables, the query is forwarded to another node and results are cached. The DUSIS plays a similar role to that of the Domain Name System (DNS) on the Web, and can be architectured in a similar way. The DNS hierarchical topology has given proofs of scalability and good performance.

4.2.2 *Services Gateway*

In a interaction between two Browserver peers when both ends are available at the same time the interaction can be direct, whereas if one end is not available, the requester must give up and retry (or not) later. In this case, the requester can subscribe to availability notification at the USIS system.

Services Gateways act as proxys to services, enabling fully asynchronous communication between entities. To enable such interaction mode, a service's description is published to a Gateway and becomes a proxied service. The Gateway, acts as a virtual service provider for that service, since the service itself remains at its provider. The service's description is parsed and a new one is created, changing all the endpoints to a Gateway endpoint that handles incoming requests. Just publishing the service is not enough though. The location of the service description document must be updated at the USIS to point at the new location, which is returned as result of publishing the service. Table 1 shows a service which description location is set to a Gateway.

The Gateway does not inspect the body content of any message, strictly restricting its action at the protocol and communication level (for message routing), thus being blind to

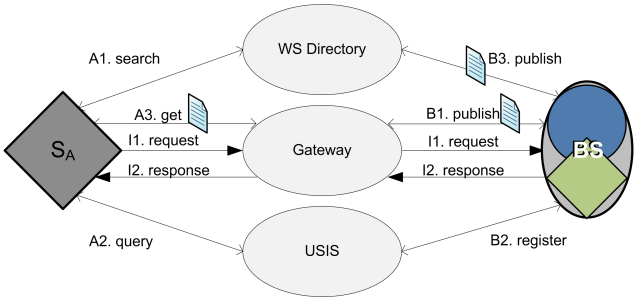

Figure 12: Service consumption through a Gateway.

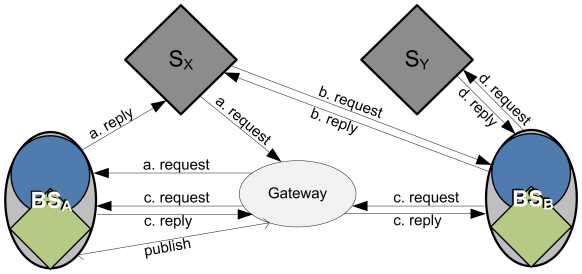

Figure 13: Interactions involving the Browserver.

the contents of the messages being exchanged. Even so, if security is critical, message contents can be encrypted and signed. The implementation here described uses Java HTTP Servlet to concretize the Gateway (based on [26]), offering a web service with *publish*, *republish* and *unpublish* operations. WS-Addressing is used in SOAP headers, defining essential parameters to enable message routing through the Gateway(s).

Figure 12 presents the consumption of a service that is proxied at a Gateway. A service of Browserver BS is published ($B1$) to the Gateway, which returns the location of the representative WSDL that is used to register ($B2$) the service description location at the USIS. To be publicly searchable, the service information is also published to the web service directory ($B3$). A server's application S_A searches for a service ($A1$) in the directory, finding that service. The USIS is queried ($A2$) for the service description document location, which is then retrieved ($A3$) from the Gateway. Given that the WSDL endpoints belong to the Gateway, the consumption of the service is done transparently through that node.

4.3 Interactions

The Browserver needs to take into account the unstable availability and unpredictable reply time of each Browserver peer. Keeping a connection open for reply to a request made to a human person is infeasible, since his presence online is volatile, any open connection might be lost, with the user's computer possibly acquiring new Internet addresses (IP) at each re-connection.

The Browserver defines another interaction mode for Browserver services. Using only one-way messages, each request defines a *replyTo* endpoint where the response is to be sent, making it unnecessary to keep an open connection or dedicated running thread for the request or reply. Although each message is unidirectional, the interaction is considered bidirectional as a related response can be expected from the requested to the requester peer.

The Gateway plays an important role to support transparent fully asynchronous communication, enabling (virtual) service consumption even in the absence of its Browserver provider. Proxying the service, a message to an unavailable peer can be sent to be later forwarded to its recipient. The response can be sent directly to the requester or through the same or any other Gateway, depending on the addressing policies contained in the message. While any non-UI service might operate in any interaction pattern, every UI service of the Browserver operates only under this asynchronous communication mode.

In Fig. 13, b is an asynchronous request-reply interaction with no intermediary nodes while b is a request an request made from BS_B to S_Y, consuming one of its web services. Interactions a and c, between server S_X and Browserver BS_A both involve a service that is proxied at the Gateway, which is set as the endpoint for service consumption. In c, the reply is sent through the Gateway while in a it is directly sent to the consumer. This is due to a request message a that defines a *ReplyTo* endpoint in S_X, with a role set to *ultimateReceiver* so the header is not processed by the Gateway and arrives unchanged at the provider. In c the role is set to *next*, so the Gateway is assumed as an endpoint for the reply.

5. BROWSERVER APPLICATION

The Browserver is suitable for simple interactions and complex workflows involving both individuals and organizations. In this section, we shortly present a workflow scenario for a business environment and show how the Browserver can be a valuable solution for it.

Scenario: SmartPrice Market (SPM) is a small supermarket with 25 employees. The inventory list is kept in a dedicated local database connected to the cash registers. Each product has a minimum inventory limit that indicates the need to re-order. The inventory management is done by the manager, Mark, who decides and makes all the orders. The orders are made to a considerable number of suppliers that might also have their own suppliers. After reaching the lower limit of a brand of bicycles, the database triggers an alert. Mark must then compile an order and send it to Two Wheels Power (TWP), a bike manufacturer and their supplier. Having a just-in-time inventory strategy, TWP needs to order parts to build the bikes. One of their suppliers is Rubber Classics (RC), to whom they send a new order of tires. Having the tires in stock, RC ships them through DHL to TWP. After receiving all the needed parts, TWP builds the bikes. The products are then shipped to SPM through DHL. During the processes, all the companies want to track the status of each order.

Figure 14 presents the workflows of this scenario, and figure 15 presents a simplified architecture for the Browserver solution.

- The infrastructure at SPM can be simplified to a product database server (which is connected to all the cash registers) and a laptop running a Browserver, with all the business services needed to manage the small supermarket;
- A remote Gateway (GW) enables asynchronous communication with SPM;
- TWP has redundant systems with business applications specifically developed to suit their needs. A server

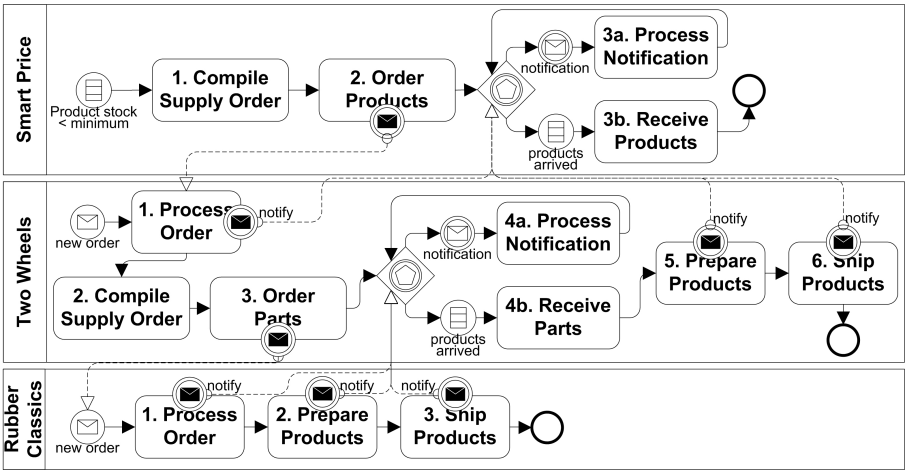

Figure 14: BPMN for the supply management scenario.

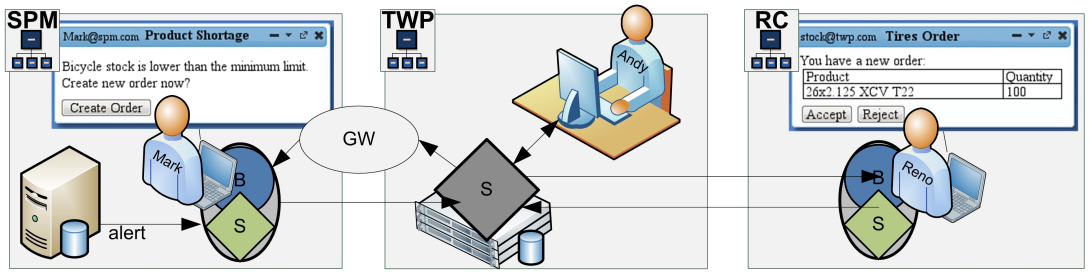

Figure 15: Simplified architectural view for the scenario..

is always available to external communication with their business partners. To interoperate with the Browserver, compliant services (considering the asynchronous Browserver model) are developed;

- RC has only simple personal computer in the managers office. The automation of processes is still at an early stage, so the Browserver is still just used to receive generic orders and notify its partners.

The order process involves the following interactions:

1. Mark's Browserver contains a service that is used by the database to alert for product shortages. This service displays a UI for mark with an alert for product shortages, giving him the possibility to create a new order;
2. Using the order service, which pre-creates all the necessary orders, Mark reviews and an sends the order to the TWP. The message indicates an *replyTo* endpoint, which location is set to a Gateway;
3. At TWP, the order service receives SPM's order and presents it to the manager, indicating the need to order new parts for the construction;
4. All the relevant notifications are sent to SPM through the notification service which is proxied at the Gateway;
5. After order approvals from Andy, orders are sent to RC's Browserver. The server at TWP is set to only request its services on work hours, as the success of the requests depend on its availability and its services are not proxyied at any Gateway;
6. At RC, Reno's browser displays the UI for the generic order service, with acceptance or rejection options;
7. Each notification is done manually by Reno, executing the local notification service for each order.

Upon receiving the notification of product expedition, either the sender queries the carrier tracking system then notifying the receiver or the receiver directly queries the carrier for status information.

6. CONCEPTUAL EVALUATION

The focus of this paper concerns the conceptual aspects of the Browserver and its differences over existing solutions. The evaluation compares the Browserver with widely used solutions, namely traditional W3C' Web Services applications and RESTful web services applications. To each conceptual aspect to evaluate, it is assigned a qualitative classification mark that indicates how well each solution suits it.

Any set of interactions between multiple entities can be seen as groups of interactions between pairs of those entities. The following are the considered conceptual aspects for interactions between a pair of entities:

(a) Parity. Measures the capacity to initiate an interaction from either of the sides and keep context (state) of that interaction, from null (none of the entities can start an interaction) to very high (both can start an interaction and maintain context);
(b) (In)Dependency. How much the relationship is dependent on a third party (external) entity, from null (always dependent) to very high (no third party is needed);
(c) Presence. How the Web presence the a person or organization is managed, from null (the presence is never guaranteed, even when the user is online) to very high

(the user's presence is always transparently guaranteed);

(d) Privacy. How private the shared information remains, from null (the information is publicly available) to very high (the information is guaranteed to be sent directly from one peer to the other. Security algorithms might also apply);

(e) Interaction patterns. How much the interaction patterns enable to model any business process between any two entities, from null (no interaction is possible) to very high (all synchronous and asynchronous interactions are possible even in the absence of the requested peer).

For the evaluation, four types of relationships and representative scenarios that occur in the Internet were identified:

Person to Person: Involves only individuals. Goes from simple message communication and file transfer to complex collaborative work. For this relationship, consider a simple scenario where a person spontaneously sends a message to another (previously unknown and with no interaction history between them) who replies;

Business to Client: Involves an organization and a client, from simple information retrievals to long running business processes. Consider a workflow with sequences of information retrievals from the client;

Business to Business: Complex relationships between two organizations with common or distinct goals (e.g. the scenario in section 5);

Client to Client: Interactions between two persons, over services offered by an organization. Consider an online auction scenario, such as Ebay.

Traditionally, a W3C WS based solution requires that the participants have specific application(s) installed so it was excluded from interactions involving individuals. However the browser is generic, universal and present in almost every user's computer, so RESTful solutions addressing the browser were considered. Solutions involving mixed approaches were not considered. Table 2 resumes the evaluation.

For person to person interaction, a typical RESTful solution would be a web site that both are using at the same time or a different web sites connected in the background that each user accesses.

In business to client, a web site is also necessary and the user is the only who can initiate the interaction. If the user does not decide to go to the web site the web site can't go to him.

In business to business, running on a dedicated infrastructure, the Browserver matches W3C WS based solutions. However, considering the scenarios described, the Browserver can overcome these. The Browserver suits simple organizational needs even if there is no dedicated IT infrastructure in the company, and can easily integrate a supply chain without the heavyweight infrastructures that traditional Web Services solutions require. RESTful web services are not widely used in this context for its lack of expressiveness, but having servers in both sides makes it possible to develop simple business processes between organizations.

In client to client, an online auction system could be developed, with a server as the auctioneer and every peer consuming its bid services. The final transaction would be done directly between the buyer and the seller. For this type of relationship, the Browserver can match the functionality of existing solutions and lower the load on enterprise servers.

7. RELATED WORK

At workflow support level, the email is still the most used tool for information workers and basic inter-organizational business support for small enterprises. Its fully asynchronous communication model, enabling a message to be sent even if its final recipient is not available at the time has been the trump that made it the universally used tool that it is nowadays. Recent works are exploring the weaknesses of the email, while taking their strengths. A platform to support business tasks, by managing its context with semantic annotations, is described in [13]. Colayer [8] and Google Wave contextualize all the communication and shared information into inter-related fully asynchronous flows of information (real time when participants are available). Both these platforms rely on application servers that can have access to the information and do give the user the capacity of providing their own services (who are restricted by the services offered to them). In contrast, the Browserver turns each user into a web service provider, while shifting from client-server to a peer-to-peer paradigm and enabling fully asynchronous communication.

Opera Unite [20] couples a browser to a server, allowing the user to share content directly with another user. However, the communication is always proxied at Opera's servers. If the user shuts down the application, his services also become unavailable. The Browserver enables pure P2P communication between entities, promoting high information privacy. The presence of it services can virtualized on remote Gateways, thus allowing to asynchronously request its services even if it is unavailable.

[16] describes a method to get the user to be seen as a web services server to a business process engine. However, the method results in sending a notification (through email or SMS) to the user, with a URL to follow. Given the browser limitation to be only a client, some external tool must be used to draw user's attention to a task. The Browserver enables requests to be sent directly to the user through the browser.

Relationships between human or organizational entities usually involve mutual information transactions with both parties shifting between provider to consumer roles, which calls for the peer-to-peer instead of the client-server paradigm. Most of the existing P2P applications are centered at file sharing (e.g., BitTorrent) or for specific collaboration purposes, such as MS Groove. The Browserver consists of a generic platform over which services representing users capabilities (like simple message communication, file sharing or complex collaborations) can be deployed to be consumed by himself and other entities.

[9] describes WSPeer as an interface to hosting and invoking web services. However, it is not aimed at user interaction but rather at creating a peer-to-peer platform for service deployment and discovery. [18] provides a means of discovering and deploying services in a structured P2P network. The services are replicated through several nodes enabling high availability, although the service location is not controllable by it's owner. The Browserver provides an automatically generated UI for executing services, as well as services for interaction with the user and does not indiscriminately replicate any of the user's services. The location of each ser-

Table 2: Comparative Conceptual Evaluation

	Parity	Independ.	Presence	Privacy	Patterns
Person to Person					
W3C' WS	—	—	—	—	—
RESTful WS	○	⌀	◐	○	○
Browserver WS	⋆	●/⋆	●	◐/●/⋆	●
Business to Client					
W3C' WS	—	—	—	—	—
RESTful WS	◐	⌀	○/◐	◐/●	○
Browserver WS	⋆	●/⋆	●	◐/●/⋆	●
Business to Business					
W3C' WS	⋆	⋆	○	●/⋆	◐
RESTful WS	⋆	⋆	○	●	○
Browserver WS	⋆	●/⋆	●	◐/●/⋆	●
Client to Client					
W3C' WS	—	—	—	—	—
RESTful WS	⌀	⌀	◐	○	○
Browserver WS	⋆	◐	●	◐/●/⋆	●

Classification: ⌀: null, ○: low, ◐: medium, ●high, ⋆: very high
—is used the concept is not applicable for the solution in such context.

vice is a decision of its owner and can be locally or remotely deployed.

[25] and [12] present methods for creating user interfaces (browser forms) from WSDL documents, enabling the user to execute the web services. Similar methods are used with the same objective by the Browserver.

8. CONCLUSIONS AND FUTURE WORK

This paper presented the Browserver, an evolution of the browser, as a human interface and as a means of empowering web users and enterprises with the capacity of not only consuming but also providing services. It contends that the unification of human and electronic services is a step forward into the Internet of Services, and that the Browserver can be applied to business relationships in an inter-organizational environment. Using the Browserver, enterprises can integrate the management of a supply chain with their partners, in a P2P fashion and with small IT infrastructures (down to only one laptop intermittently connected to the internet). Used either as a personal or as an enterprise application platform, the Browserver offers:

- A generic application platform, suitable for full workflow and collaboration support.
- Information privacy and independence form third party service providers. The Browserver application can operate without the need for intermediaries if it as dedicated infrastructure (a computer always available), or with services Gateways that are application agnostic and information blind.
- A complete service paradigm for user interaction. The user can be directly invoked as a web service, and no external means (e.g. email) are necessary to draw user's attention.
- Interactions with users can be proactive and not only reactive to user's actions.
- Improving costumer intimacy and the opportunity to develop new business models with clients that are able to interact with the organization's applications in a peer-to-peer paradigm.
- Offline work capabilities, by having the needed services and resources for an application executing at the local server.

As result of a wide scale adoption (a generalized use, as the browser), the Browserver is expected to have an extreme impact. The email and other communication platforms will certainly become accessory and not mandatory for communications and interactions involving persons in the Web. Designing and executing workflows benefits from the existence of a single universal platform enabling the composition of multiple different tasks. Sending files, making requests, chatting, and so on, all can be done with the Browserver, directly between the interested parties. User agents (web services) automate decisions such as personalized email replies and conciliating agendas for meetings and work planning.

Social networks can be built without central servers. A group of friends is maintained by keeping each friend's URN in each Browserver. Publishing services in such networks can promote new collaborations and spontaneous content (resources) creation. New jobs can arise by publicizing one's capabilities in the Web, and by having enterprises looking for the most suitable person to each task.

The architectural solutions presented here have already been implemented and tested. Service delegation is another concept that can play an important role in the continuous presence (or availability) of the services and constitutes work in progress.

- Still as a work in progress, the Unified Services Model [5] and the Unified Services Implementation Language (SIL) represent a perfect fit for the Browserver. Compare a SIL implementation of the architecture with the one here presented.
- Address an existing solution or design a new one for message routing through NATs.
- Study and test distributed solutions for high availability and performance of Gateway.
- Service delegation for continuous presence (availability) of the services. It consists on transferring the full service implementation to a remote trusted entity capable of executing that service in behalf of the delegator (always ensuring the owner's control over the location of the service) in a secure way and granting all the information privacy necessary.
- Determine the full set of primitive services and opera-

tions for an operational basic Browserver in heterogeneous environments.

- Usability of the generated user interfaces must also be object of depth study, as this is a crucial subject for the success of the Browserver adoption.

9. REFERENCES

[1] T. Bellwood, S. Capell, L. Clement, J. Colgrave, M. J. Dovey, D. Feygin, A. Hately, R. Kochman, P. Macias, M. Novotny, M. Paolucci, C. von Riegen, T. Rogers, K. Sycara, P. Wenzel, and Z. Wu. Uddi version 3.0.2, October 2004. www.oasis-open.org/committees/uddi-spec, last access on 2010-10-14..

[2] T. Berners-Lee, R. Fielding, and L. Masinter. Rfc 3986, uniform resource identifier (uri): Generic syntax. Request For Comments (RFC), 2005. http://www.ietf.org/rfc/rfc3986.txt, last access on 2010-10-14.

[3] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web service definition language (wsdl). Technical report, March 2001. http://www.w3.org/TR/wsdl, last access on 2010-10-14.

[4] D. Crane and P. McCarthy. *Comet and Reverse Ajax: The Next-Generation Ajax 2.0*. Apress, Berkely, CA, USA, 2008.

[5] J. Delgado. Modelação de sistemas com serviços unificados. unpublished report, https://dspace.ist.utl.pt/bitstream/2295/581353/1/servicos.pdf, last access on 2010-10-14., 2008.

[6] Z. Du, J. Huai, and Y. Liu. Ad-uddi: An active and distributed service registry. In *In C. Bussler and M.-C. Shan, editors, 6th VLDB Int'l Wksp. on Technologies for E-Services, volume 3811 of LNCS*, pages 58–71. Springer, 2006.

[7] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[8] C. GmbH. Colayer. http://www.colayer.com. http://www.colayer.com, last access on 2010-10-14.

[9] A. Harrison and I. Taylor. Dynamic web service deployment using wspeer. In *Proceedings of 13th Annual Mardi Gras Conference - Frontiers of Grid Applications and Technologies*, pages 11–16. Louisiana State University, February 2005.

[10] I. Hickson. The websocket API. W3C Working Draft, June 2010. http://dev.w3.org/html5/websockets/, last access on 2010-10-14.

[11] I. Hickson. The websocket protocol. IEFT Draft, May 2010. http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-00, last access on 2010-10-14.

[12] M. Kassoff, D. Kato, and W. Mohsin. Creating guis for web services. *IEEE Internet Computing*, 7:66–73, 2003.

[13] M. Laclavik, M. Seleng, E. Gatial, and L. Hluchy. Future email services and applications. In *Proceedings of the Poster and Demonstration Paper Track of the 1st Future Internet Symposium (FIS'08)*, volume 399, pages 33–35, 2008.

[14] A. Lantz. Does the use of e-mail change over time. *International Journal of Human-Computer Interaction*, 15:419–431, June 2003.

[15] D. Marginian and J. Walke. Direct web remoting - easy ajax forjava, 2009. http://directwebremoting.org, last access on 2010-10-14.

[16] D. Matsubara and K. Miki. Method and apparatus for peer-to-peer access. Patent US 2004/0148434 A1, Jul 2004.

[17] F. McCabe, D. Booth, C. Ferris, D. Orchard, M. Champion, E. Newcomer, and H. Haas. Web services architecture, February 2004. http://www.w3.org/TR/ws-arch/, last access on 2010-10-14.

[18] R. Mondejar, P. Garcia, C. Pairot, and A. F. Gomez Skarmeta. Enabling wide-area service oriented architecture through the p2pweb model. In *WETICE '06: Proceedings of the 15th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 89–94, Washington, DC, USA, 2006. IEEE Computer Society.

[19] H. F. Nielsen, N. Mendelsohn, J. J. Moreau, M. Gudgin, and M. Hadley. SOAP version 1.2 part 1: Messaging framework. W3C recommendation, W3C, June 2003. www.w3.org/TR/soap12-part1/, last access on 2010-10-14.

[20] Opera. Opera unite website. http://unite.opera.com/, last access on 2010-10-14.

[21] J. Resig. jquery: The write less, do more, javascript library, 2010. http://jquery.com, last access on 2010-10-14.

[22] L. Richardson and S. Ruby. *Restful web services*. O'Reilly Media, 2007.

[23] C. Schroth and T. Janner. Web 2.0 and soa: Converging concepts enabling the internet of services. *IT Professional*, 9:36–41, 2007.

[24] M. Schumacher, T. van Pelt, I. Constantinescu, A. de Oliveira e Sousa, and B. Faltings. Federated directories of semantic web services. In *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, pages 368–369, New York, NY, USA, 2007. ACM.

[25] K. Song and K. H. Lee. An automated generation of xforms interfaces for web services. *IEEE International Conference on Web Services*, 0:856–863, 2007.

[26] C. Venkatapathy and S. Holdsworth. An introduction to web services gateway, May 2002. http://www.ibm.com/developerworks/webservices/library/ws-gateway/, last access on 2010-10-14.

[27] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. F. Ferguson. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[28] S. Whittaker and C. Sidner. Email overload: exploring personal information management of email. In *CHI '96: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 276–283, New York, NY, USA, 1996. ACM.