INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

The Browserver

Miguel Fernando Evaristo Raposo

Dissertation for the Degree of Master in

Information Systems and Computer Engineering

Jury

President: Prof. António Rito Silva
Supervisor: Prof. José Delgado
Examiner: Prof. Bruno Martins

October 2010

Acknowledgements

This dissertation is culmination of a long period of my life, and the final stage of an important step towards the future. I certainly could not have been completed without the support, friendship and love from the ones that I consider closer to me.

During the last few years I have had less time to myself, to my girlfriend, to my family and to my friends then the time I would like to have had. In the road to the MsC, I've not only dedicated many days and nights to the University, but had also to multiply myself during the tree years I've worked at night and weekends to grant enough income for the expenses. Nevertheless, I managed to have great persons around me, that were very supportive, comprehensive and enabled that everything went well. The last months were dedicated to this dissertation, and even though their contributions were not technical, just knowing they have always been there all along the way has been crucial.

I would like to start thanking my supervisor Prof. Dr. José Delgado, for the opportunity of writing this dissertation, for the attention demonstrated and the support he offered me. I'm grateful for the numerous revisions he made to my writings, the praise and the incitement to produce more. I'm thankful for the time he expended with me, discussing what mattered the most to my work, while keeping me on the right path.

As for my friends, I would refer David Lucas as the colleague with whom I worked the most, the one who spend the most time with me in the last years. and who as been a really fantastic mate. I also include Daniel Mendes, Joo Oliveira and Daniel Pando as extremely important, as they have crossed the bridge from the start with me. I am certain that the last six or seven years were just the beginning.

To my childhood friends Carlos Costa and Rodolfo Botelho who provided and still provide a lot of weekend fun and to my room mate Vitor Lampreia getting into the University would be even harder and they were three great years, I would like to give you all a great hug.

I am truly grateful to my mother, Maria Evaristo, for her dedication on raising me, for the education she gave me, for all the care she provided me, for all the love that she has for me, for the proud she feels for me. Even though I feel difficult to externalize my feelings, they are truly reciprocal for her as she is extremely important in my life, and I hope she will be always there as I will always be there for her. For my father, Fernando Raposo, I know that even though he does not demonstrate he his proud of me. Thank you both for always giving me all the conditions I needed, thank you both for making me who I am.

Last, but certainly not least, I would like to thank my beloved girlfriend, my passion, my more then

anything, Sofia Neves. Although the five years and eight months (at the time of this writing) are short in each one's life, our actual life is way shorter then the common life I hope we will have. With you, my life as changed, and keeps changing every day and we change with each other, and we love each other more every day. Thank you for all the patience, for all the comprehension, thank you for all the support, thank you for for being there, thank you for being with me.

The path through here has not been easy, but I have covered it and I feel proud of what I have achieved. I hope this is just the beginning of a new and longer path to traverse and I hope I always have my friends and family with me as I will be with all of them. For all the select group I acknowledged here I desire only the best, as they all deserve the best.

Abstract

Web applications are now built on the principle that users interact with them through a generic, universal browser. However, the client-server paradigm defines the client as the sole entity with real initiative capacity, being the server a passive entity in the relationship. Also, server-centric applications often do not guarantee information privacy resulting in reluctance in its usage.

Although we already perform services in the Web, we are not seen as service providers, as we are still mere consumers of services and resources that other entities provide (e.g. Google apps). We are still unable to publish and offer our own capabilities in the Web and that is due to the lack of interoperability and a generic and universally used user interface for services to unify human and electronic services.

This dissertation presents the Browserver (which includes a browser and a server) as a human interface for web services and a generic universal platform for service provisioning. 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 solution focus on changing from client-server to a peer-to-peer (P2P) paradigm, enabling direct interactions between entities and extinguishing the need for applicational intermediaries. An architectural solution and an implementation approach existing technologies are described. The dissertation further presents business scenarios and compares 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**

Resumo

As aplicações Web de hoje são desenvolvidas sobre o princípio de que os utilizadores podem interagir com estas através de um genérico e universal *browser*. No entanto, o paradigma cliente-servidor define o cliente como a única entidade com capacidade de iniciativa real, desempenhando o servidor um papel passivo na relação. Além disso, aplicações centradas no servidor frequentemente não garantem a privacidade da informação, o que resulta em relutância na sua utilização.

Apesar de já realizarmos serviços na Web, ainda não somos vistos como fornecedores de serviços, sendo apenas meros consumidores de serviços e recursos que outras entidades disponibilizam (e.g. Google apps). Ainda somos incapazes de publicar e oferecer as nossas próprias capacidades na Web, devendo-se isso à falta de interoperabilidade e de uma interface de utilizador genérica e universal, para a unificação de serviços humanos e electrónicos.

Esta dissertação incide sobre o Browserver (que inclui um browser e um servidor) como interface humana para serviços e plataforma genérica e universal para fornecimento de serviços. A capacitação de indivíduos e PMEs com uma plataforma de baixo custo para suporte a *workflows* que não requer infraestruturas dispendiosas, garantindo a privacidade da informação e minimizando a dependencia de terceiros é o objectivo do Browserver. É focada a mudança de paradigma de cliente-servidor para *peer-to-peer* (P2P), possibilitando interacções directas entre entidades sem necessidade de intermediarios aplicacionais. Uma solução arquitectural e uma implementação utilizando tecnologias existentes são descritas e so apresentados alguns cenários, comparando o uso do Browserver face a soluçoes tradicionais para interacções inter-organizacionais e entre empresas e clientes.

Palavras-chave: **Serviços Electrónicos, Serviços Humanos, Interface de Utilizador, *Browser*, Servidor, *Peer-to-Peer*, *Workflow*, Colaboração**

Contents

List of Figures

List of Tables

List of Listings

List of Acronyms

AJAX Asynchronous JavaScript and XML

API Application Programming Interface

BPMN Business Process Modelling Notation

CSS Cascading Style Sheets

DNS Domain Name System

DOM Domain Object Model

DUSIS Distributed Universal Service Identifier System

DWR Direct Web Remoting

IoS Internet of Services

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

JAX-WS Java API for XML Web Services

JSON JavaScript Object Notation

NAT Network Address Translation

OWL-S Web Ontology Language for Services

P2P Peer-to-Peer

PDA Personal Digital Assistant

REST Representational State Transfer

SIL Unifed Services Implementation Language

SOA Service Oriented Architecture

SOAP Simple Object Access Protocol

UI User Interface

UDDI Universal Description, Discovery and Integration

URI Uniform Resource Identifier

URL Uniform Resource Locator

USI Universal Service Identifier

USIS Universal Service Identifier System

W3C World Wide Web Consortium

WOA Web Oriented Architecture

WS Web Service

WSDL Web Service Description Language

XML Extensible Markup Language

XSLT Extensible Stylesheet Language Transformations

1. Introduction

1.1 Motivation

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.

With services already being the main paradigm at enterprise integration and the Internet of Services (IoS) [Schroth 07] 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. REST [Richardson 07] based web service applications are developed knowing that they cannot interact bilaterally with the user. Pages are all pre-built by the application and AJAX [Crane 08] is used to minimize communications, simulate requests from the server, and more interactive user interfaces (UIs). On the other hand, SOAP [Weerawarana 05] based web services applications require specific clients and are not compatible with the universal generic client (the browser) that every user has.

With such limitations, human centered processes are hard to design and execute, as they typically rely on different tools and communication methods to involve the user. If a user is involved in some business process, there is typically no way to directly interact with him, and one of the typical solutions to draw his attention is the email (universally used, as the browser is), by either

requesting a reply to a message or to follow a link to some Web application (through a web page). The email is nowadays the most used tool to contact and request someones services, having become an indispensable tool for both individuals and enterprises with its usages greatly overcoming its original intent [Whittaker 96, Lantz 03].

As Web applications (for communication, file sharing, collaboration, ...) are mainly server-centric (including the email), requiring all communications to go through a central server (or several distributed applicational servers), even if the user is just interacting with another user, the information is made available to more entities than those directly involved and its ownership becomes shared and the usage given to it is often unknown, therefore raising privacy issues. Users (or an agent on their behalf) should be able to provide electronic services themselves and be first class peers in web interactions and business or generic processes without the need for applicational intermediaries.

In sum, the main motivations that drive the Browserver are:

- 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.

1.2 The Browserver Solution

The Browserver is conceptualized as a natural step from the browser, a simple generic client based on Web standards, to a generic and universal web services peer (both capable of consuming and providing services) enabling a personal presence in the Internet of Services by unifying human and electronic services. It enables a user to have its own services, representative of the capabilities he wants to expose to the world (public services), as well as private services for own consumption. It can be used as a personal as well as an enterprise tool, as shown in this document.

Much like the browser has been an evolution from specific clients to a generic one, the Browserver extends the user capabilities in the Web from a simple navigator to a service provider. In short, the Browserver can be seen as:

- A generic, universal browser. A browser capable of rendering and executing existing Web browser technologies;
- A generic, universal server. An applicational server united with the browser enabling the provision of services to the Web;
- Peer-to-peer (P2P) interaction model. Contrasting with the restrictive client-server model, services and applications can be conceived knowing that any entity in the network can make requests and reply to requests.

Direct, P2P interactions become the norm instead of having to resort to centralized application servers for user interactions, entailing a paradigm change for web usage, from client-server to peer-to-peer, and not just for file sharing. Applications such as email, instant messaging, social networks, collaborative document edition and workflow systems can be implemented without necessarily depending on some central server system.

Relationships between human or organizational entities usually involve mutual information transactions with both parties shifting between provider to consumer roles. We consider this the relationship parity, and this a concept that does not fit the client-server paradigm since only the client has initiative to start an interaction (limiting him to be a consumer). To achieve parity in business or personal relationships in the web, peer-to-peer is the necessary paradigm.

Each peer has a unique identifier, which is the base part of each of its services. The Universal Service Identifier (USI) enables univocal identification of each service while associating it with its provider. Each service can be either private or public. If defined as public, it will be registered at a distributed Universal Service Identifier System (USIS) that enables queries regarding services, like their location (or their description document) or the services offered by a particular peer.

To enable service continuity even in the absence of its owner (and the browserver where the service is deployed), Gateway nodes act as service proxies, receiving, holding and forwarding requests, ensuring its delivery and enabling a fully asynchronous interaction mode. The Gateways do not act as application level nodes, only working as communication facilitators and restricting its action to message headers for routing.

1.3 Objectives and Contributions

The main objective of this dissertation is to show how the Browserver can benefit individuals and enterprises by empowering each user with service provisioning capabilities and decreasing the need for centralized servers for collaboration and workflows involving persons.

To achieve its main objective, this document:

- conceptualizes the Browserver;
- describes the Browserver architecture;
- describes an high-level prototype implementation as a proof of concept for the Browserver;
- presents concrete scenarios, applying a browserver solution for them;
- conceptually evaluates Browserver-based solutions over traditional solutions;

It is shown how the Browserver architecturally fits both pure P2P interactions as well as current client-server services architectures, with intermediate architectures using P2P as a basis but maintain a central server presence if desired. This allows a gradual change between scenarios and a higher level of agility in adapting to the highly variable environment which small enterprises must survive in. The benefits of the Browserver as a platform for the Internet of Services are highlighted.

This dissertation defends the Browserver as the tool for each Web user instead of the browser and shows how the Browserver can improve our presence in the Web, our interactions with other persons and our integration in business processes, while enabling:

- Individual service provisioning, exposing our capabilities in the Web.
- Control over information, which stays in our own private computers instead of remote servers.
- Privacy, as information flows can occur without intermediaries who can read and use the information.
- Improved interactivity with web services and business processes, through a generic user interface.

The following papers resulted from the work performed in this dissertation:

- Miguel Raposo and José Delgado. Bridging The Browser And The Server. In Proceedings of the Informatics Symposium (INForum) 2010, pp 587 - 898, Braga, Portugal, September 2010.
- Miguel Raposo and José Delgado. Empowering The Web User With A Browserver. In Proceedings of the Conference on ENTERprise Information Systems (CENTERIS) 2010, Viana do Castelo, Portugal, October 2010. Springer. (accepted for publication).

1.4 Outline

In the rest of the document the Browser is conceptualized in Chapter 3, Chapters 4 and 5 contain the architecture and implementation descriptions, respectively, while Chapter 6 presents business scenarios along with Browserver solutions for them. Chapter 7 presents the conceptual evaluation and the related work is presented in Chapter 8. Finally, Chapter 9 draws the relevant conclusions and discusses the future work.

2. Background

This chapter presents the background from which the Browserver appears, with the evolution of the Web and relevant technologies associated being briefly presented. The service paradigm and peer-to-peer architectures are described.

2.1 World Wide Web

Until the early 1990's, Internet applications were made with specific client and server side components (Figure 2.1) and specific protocols with interactions limited by the existence of the necessary application specific client on each user's machine.

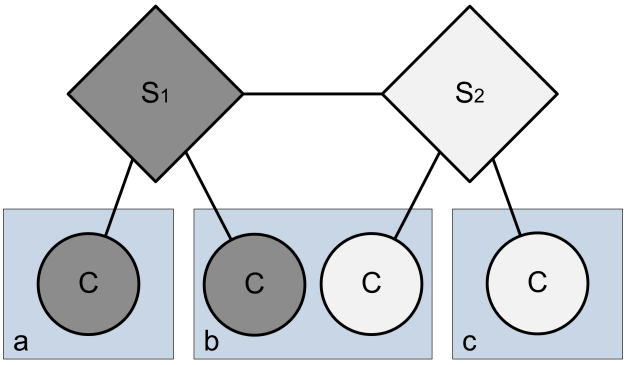

Figure 2.1: A Specific client for each specific application.

In the early 90's [Berners-Lee 94b], the World Wide Web (Web) was developed to enable knowledge sharing among distant persons, through standardized communication languages and transfer protocols. The Web has defined its basic model over the client-server architectural topology, with:

- The URI (Universal Resource Identifier) [Berners-Lee 05] which enables universal addressing in the network. The URI defines a unique identification scheme for every resource on the Internet, enabling its location for retrieval.
- The HTML (Hypertext Markup Language) [Jacobs 99], a common basic language of interchange for hypertext. It is defined to be a language of communication which defines a standardized way of structuring information and is independent from the transfer protocol.
- The HTTP (Hypertext Transfer Protocol) [W3C 09], a protocol for efficient transference of information, enabling hypertext jumps (using hyperlinks) and resource retrieval. It is aimed

at transferring any resource on the Web, not being limited to HTML documents.

From its beginning the Web as grown from inside private organizations to the universal usage that we see nowadays. During its first decade, it was mostly seen as a repository, through which some entities could spread information to others, who could search and consume it. The Web user was seen primarily as an information consumer and collaboration was still mostly restricted to organizational entities that controlled the Web servers, and the resources that were made available.

The client-server paradigm was imperative in Web interactions, with the server playing a mere reactive role to requests made from a client, and never having the initiative to contact the client neither saving any state on the communication, which limited the interaction level with persons on the Web.

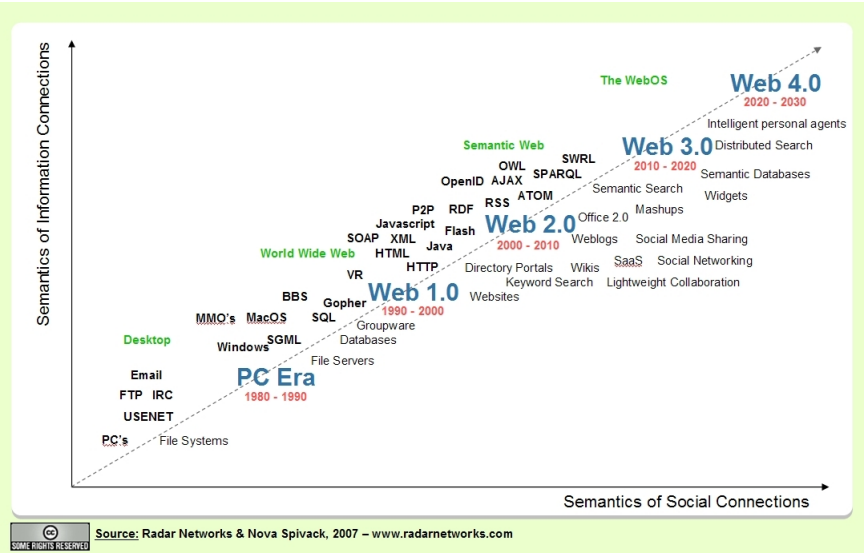

Figure 2.2: Evolution of the Web.[1]

The Web 2.0 [Hoegg 06] puts the user as an active information generator by providing him collaborative and social platforms along with many other tools to support his workflow. The Web 2.0 [O'Reilly 07] sets the network as a platform for applications and services remotely provided to the client, promoting the Software as a Service (e.g. Google Apps).

To enable such philosophy, technologies that enhanced the interactivity with the user (section 2.1.2), extended the basic standardized Web model, empowering developers with new tools for new provider to consumer interactions. However, the client-server is still the basic Web model, with a clear separation between the client, essentially a human interface, and the server, centrally located in a logical hub-and-spokes architecture. Information remains stored at the server side, with the user (the actual information owner) having little or no control over it. The information is

[1] Nova Spivack, Radar Networks, www.radarnetworks.com; www.twine.com

co-owned by the entity that provides the services and therefore privacy issues still constrain their usage.

The predicted future Web 3.0 (Figure 2.2) introduces a greater concern over the semantics of the information. The aim of the semantic Web will be to give more value to the contents with the introduction of intelligent systems that interpret the information, manage the knowledge and have methods to turn all that information more useful and significative for the Web user [Lassila 07].

Although, in this vision, even with new standard technologies the architectural paradigms on the Web and the interactions between persons and between these and the services tend to be the same as with the actual Web, even with increased privacy issues.

2.1.1 Web Servers and Web Browsers

A Web server is a computer application that enables the delivery of Web resources to Web client applications using the HTTP protocol. Web servers can run on environments like large dedicated computer servers or a simple personal computer. The Web browser is also a computer application that plays the role of client for the Web. Its main function is to retrieve, present and navigate through resources, acting as a user interface for the Web.

The Web server and the Web browser enabled a single client for many applications, breaking the growth on the number of clients for the increasing number of Internet applications that are now delivered over Web technologies. Nowadays, the browser is a standardized universally used generic client and user interface for almost any web (and even local) application (Figure 2.3). It enabled the change to an era where the browser is the *de facto* standard for the human to computer interface, especially in the case of distributed and collaborative applications.

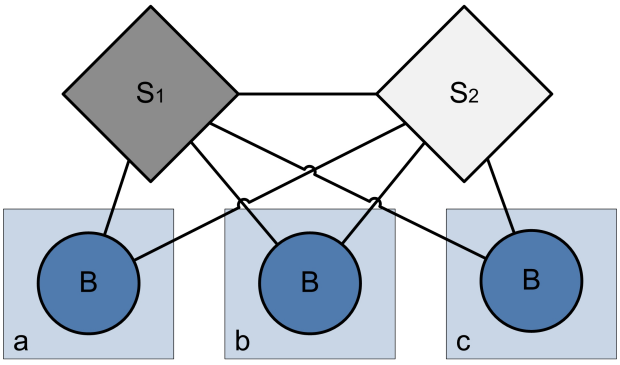

Figure 2.3: The Web browser, a universal client for Web applications.

Although almost any platform can run a Web server, its usage has been associated with organizations that are sharing their information with the world (or internally), publicizing their services or providing services to others. Web servers can be either dedicated to a specific Web appli-

cation for better performance or shared between multiple application. Replicated and distributed architectures can guarantee the reliability and availability of Web applications. However, for the information (or resource, or service) to be available, the server must be running and be accessible on the Web. Since a typical Web user does not have a continuous presence on the Internet (he is not always online), he typically does not run a server on his side, but rather only the Web browser.

Nowadays, any individual can rent a dedicated or shared Web server to provide his contents to the Web, that way passing to the renter the responsibility of having it always available. However, the cost it carries and security and privacy issues still remain as barriers for the deployment of sensitive information on such servers.

The coupling of a browser with a server has already been initiated by Opera, with the Opera Unite. Resources can stay at the user's computer, and be made available to the Web, however, the communication is always proxied at Opera's servers (who acts as intermediary for any interaction), and if the user shuts down the application, all his resources become unavailable.

2.1.2 Extensions to the Base Model and Emerging Technologies

The Web was initially designed to share knowledge among persons working in distant places but it rapidly developed to be a way to publicize products, brands and organizations, and to be a platform for service provisioning.

Since the client-server paradigm does not enable a Web server to make requests to a Web browser, and the simplicity of the HTTP together with the lack of expressiveness of HTML do not enable complex and richer interactions with the Web users, new technologies were introduced to complement the basic Web scheme. The static nature of Web pages and the lack of interactivity of the standardized protocols promoted the appearance of new technologies that extended the basic Web model and enabled richer and more interactive Web applications. The browser was given more responsibilities, by migrating part of the information processing and even part of the business layer of a Web application from the server to the client side. Some of the most relevant technologies are:

- Java applets, which are small compiled Java applications (or simple blocks of code) that can be embedded in Web pages to perform a task (e.g. accessing the file system). Java applets can run nearly on any platform and browser, provided that the computer has the Java Virtual Machine necessary. Nowadays, their usage is limited by the browsers for security reasons.
- Javascript [Flanagan 06], is an browser-interpreted scripting language that is embedded in HTML pages and interact with its Document Object Model (DOM) (e.g. for form validations), enhancing the user interfaces presented in the browser.

- AJAX (Asynchronous Javascript and XML) [Garrett 05], which enables web applications on the client side to retrieve data from the server asynchronously (through the XMLHttpRequest [van Kesteren 09] interface) in the background without interfering with the display and behavior of the existing page, leading to more dynamic web pages.
- Comet and Reverse-Ajax [Crane 08] help to overcome the limitations of the client-server model, enabling a server to actively make requests to the user. Comet refers to long-lived HTTP connections, enabling low-latency communication between browser and server. Reverse-Ajax uses continuous polling from the client to the server for changes or server pushing to the client using Comet connections enabling a server to send data to the client without it without having been explicitly requested.

The appearance of this technologies solved some base standards limitations, but they consist of glue for the gaps left open on the base definition of the Web, and that were seen as a necessity by the developers. HTML5 [Hickson 10c] and Web Sockets [Hickson 10a, Hickson 10b] are already being proposed as standards to enhance the Web.

The HTML5 draft standardizes the interactivity directly in HTML, with dynamic user interface creation and new and more powerful tags and controls. Dynamic content such as video and features like drag-and-drop are now becoming part of the standard and not dependent of a plug-in extension.

Web Sockets promise real bi-directional connections between the browser and the server, enabling better and faster communication between browser and server than AJAX. However, the first interaction will always be from the client, and if he leaves the pages related to the server the relationship ends.

2.1.3 Collaborative Platforms

The Web provides a great number of tools for collaboration purposes. Blogs, forums and wikis have been pioneers of collaboration through the Web. Along with the evolution of technology, the applications that are offered to the user in the Web are increasing. Today persons are able to easily collaborate through the Web, using applications such as:

- Zoho[2] and Google Docs[3] which offer collaborative document sharing and editing;
- Google Groups[4] , enabling group threaded conversations with search functionalities and filtered delivery through email;

[2]Zoho: http://www.zoho.com/, last access on 2010-10-05
[3]Google Docs: http://docs.google.com/, last access on 2010-10-05
[4]Google Groups: http://www.groups.google.com/, last access on 2010-10-05

- Colayer[5] and Google Wave[6], which integrate many different systems in one, enabling real time collaboration (with chat, document editing, between numerous people;
- Twitter[7], a micro blogging application that enables real-time communication with a great number of persons at the same time;
- Doodle[8], a facilitator that eases the scheduling of group meetings or voting for a choice between many alternatives.

The Web as developed to be a great platform for collaboration, however, the user gets to be more and more dependent from a third parties and from the availability they can grant from the applications. People get to be hostages of the good will and good management of an organization that controls applications that can become crucial for the user. Privacy becomes compromised, has the user does not know who really has access to his information.

2.2 Email

The email has its roots almost twenty years before the appearance of the Web. However, the large scale adoption of Internet email as a global standard began in the early 90's, and since then its usage for collaboration and personal organization purposes as greatly grown, with the increase of Internet users.

The email success is mainly 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.

Nevertheless, email overload [Whittaker 96] has been a problem that has not shown signs of slowing down [Lantz 03]. The usage purposes of the email surpass its basic original intent, which was simply to send information from one end to another. Besides that, the information that flows through the email is unstructured, and requires that people spend much time analyzing it. Information contextualization is not an implicit feature of the email and it depends on each user to maintain the context (the whole previous conversation on some subject) during successive message exchanges (which is critical on temporally sparse interactions). Even more, central servers

[5] Colayer: http://www.colayer.com/, last access on 2010-10-05
[6] Google Wave: http://wave.google.com/, last access on 2010-10-05
[7] Twitter: http://www.twitter.com/, last access on 2010-10-05
[8] Doodle: http://www.doodle.com/, last access on 2010-10-05

are used to store all the emails sent and received, with the number of entities that have access to the information being unknown to the main interested parties.

2.3 Services

In this dissertation, 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. Numerous services can be composed into a new service provided by one or more entities, that can be used to create a new service by composition with other services, enabling more complex services at the expense of services of lower granularity.

Services can be either human or non-human (electronic, physical resources). A Web service is a a service that is electronically provided and consumed in the Web and is necessarily not attached to any specific set of technologies. However, the term Web Service has been coined to be used with a set of technologies standardized by the W3 Consortium and described in [Erl 05]. When referring to any service provided in the Web the term *Web services* is used, while *Web Services* (WS) (capitalized) refer to the W3C standardized form of Web services.

W3C Web Services provide a set of technologies that can be used to implement Service Oriented Architectures (SOA) while RESTful Web services and Web Oriented Architectures (WOA) [Hinchcliffe 08] are also starting to get more relevance in the Internet of Services (IoS) [Schroth 07]. Both have strengths and flaws being the ideal services somewhere in between them. To fit that place, the Unified Services Model is an in-progress proposal for an new approach for the service paradigm, including both a framework [Delgado 08a] for modeling services built from scratch and the technologies [Delgado 08b] to electronically implement them.

2.3.1 Service Oriented Architecture

The actual Service Oriented Architectures refer to system design and development over the service paradigm that follow a set of design principles:

- Standardized Service Contract: services agree on communication via standard service-description document(s);
- Loose Coupling: service relationships minimize inter-dependencies;

- Abstraction: the service logic is not exposed, only its interface is known to the outside world via the service contract;
- Reusability: logic is divided into services that promote reuse;
- Autonomy: services have significant degree of control over their encapsulated logic and resources;
- Statelessness: to promote availability and scalability, services minimize or defer state management tasks;
- Discoverability: services are supplemented with descriptive meta-data and the systems were the services operate provide mechanisms for their discovery and interpretation;
- Composability: services can be participants in compositions that result into new and more complex services.

2.3.1.1 Web Services

Web Services [McCabe 04, Weerawarana 05] have been standardized by the World Wide Web Consortium (W3C), and consist of a set of technologies that follow the principles defined for SOAs. Each service is identified by an URI and described through XML artifacts, namely the Web Services Definition Language [Christensen 01] which enables the service contract.

The communication is message oriented and uses the Simple Object Access Protocol (SOAP) [Nielsen 03] , relying on XML, for exchanging structured information independently of the transport protocol. HTTP is the most common protocol on which Web services operate, although, at enterprise level, message-oriented middleware such as the Java Message Service (JMS) [Hapner 02] plays an important role for granted message delivery and enterprise integration.

The Universal Description, Discovery and Integration [Bellwood 04] system provides register and search mechanisms for services, enabling service discoverability. UDDI includes service information as address, contact and identification of its provider (white pages), categorization based on standard business taxonomies (yellow pages) and technical information about the services (green pages). The UDDI system is traditionally centralized (or replicated) although, Ad-UDDI [Du 06], has proven its feasibility on transposing its concepts to an active distributed environment.

Web Services technologies have been extended with numerous additions which form the WS-* extensions family for WS. The WS-Security [B. Atkinson 02], for message signing and encryption for SOAP and WS-Reliability [Iwasa 04] for reliable communication over HTTP are just two of those. At semantic level, there are also extensions such as OWL-S [Martin 04], which proposes

an ontology for semantic description of services which allows a greater automation on discovery, invocation, composition and monitoring services and resources.

Being consistent with the SOA principles, the Web Services appear as a preferred technology for new inter-organizational integrations. However, they present an high complexity and low performance due to the numerous layers that compose it and the high verbosity of XML (its base technology).

2.3.2 Web Oriented Architecture

The term Web Oriented Architectures (WOA) [Hinchcliffe 08] relies on the concept of Resource Oriented Architecture (ROA) [Richardson 07] and Representational State Transfer (REST) [Fielding 00]. Its architectural principles are:

- Resources: all the information is represented by resources in the Internet (documents, web pages, media, etc). The resource format must be based in standard technologies consumable by any canonical Web client.

- Resource Identification: each resource must be unequivocally identified through an URI, which must enable its localization in the network. The URI must be self-descriptive, enabling the identification of the type of resource so that the Web client knows how to consume it.

- Access to Resources: the access to resources must not require more knowledge than the operation on the Internet.

- Resource Manipulation: resource manipulation is exclusively made through Web components (browsers and servers). It is their responsibility to understand the representations and validate the transition states of the resources they manipulate.

- Contract: the service contract is implicit in the resource representation that the consumer receives.

- Link: resources Web contain themselves URIs to other resources, enabling the creation of a larger network.

Figure 2.4 presents a vision over WOAs, in which the resources are represented by hypertext, which have links to other resources and are manipulated by Web components through Web technologies.

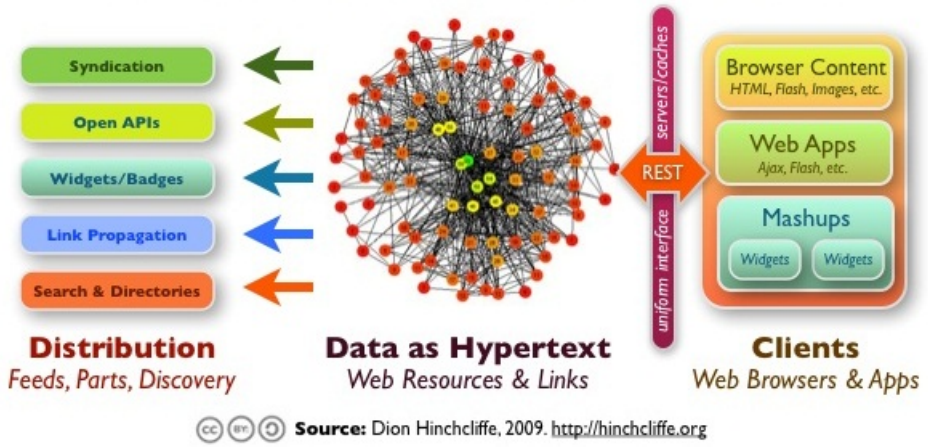

Figure 2.4: Web Oriented Architecture View [Hinchcliffe 08].

2.3.2.1 RESTful Web Services

REST (Representational State Transfer) [Fielding 00] is an architectural style for Web development. The aim is the scalability and performance of Web applications, using generic interfaces and promoting stateless interactions by transferring representations of resources based on their identifier (URI), instead of operating directly over those resources. The model is client-server and promotes the intermediate caching of resources. The resource identifier and its representation must contain enough information for its manipulation by the client, using only HTTP operations GET, POST, PUT and DELETE which semantically correspond to SQL operations SELECT, INSERT, UPDATE and DELETE.

RESTful Web services [Richardson 07] refer to the design of systems based on a service paradigm (services in the form of resources) and strict utilization of REST technologies. When compared to Web Services, RESTful web services need less computational resources and having higher performance and being highly scalable.

2.3.3 Security

To ensure Web services security, various technological approaches are available. Given that the HTTP communication protocol in the Web, to ensure security of flows of information, the implementation of this over SSL/TLS cryptographic mechanisms (HTTPS [Rescorla 00]) provides independently applicable security services for transaction confidentiality, authenticity/integrity and non-repudiability of origin. Nevertheless, additional options exist. For W3C Web Services, the WS-Security [B. Atkinson 02] extends the communication protocol for securing messages, while for the XAdES specification allows digital signing of a XML document.

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 (Figure 2.5), peer-to-peer systems enable direct interactions between its participants (Figure 2.6). 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.

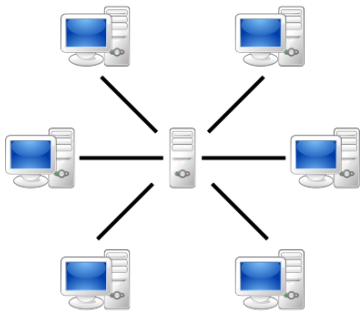

Figure 2.5: Centralized server-based system.

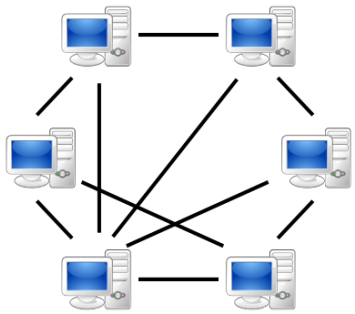

Figure 2.6: Decentralized Peer-to-Peer system.

Peer-to-peer architectures have two main characteristics that define the way the system operates:

- Structure:
  - Structured: There are rules for node positioning as well as for content (resources) distribution in the network. Distributed Hash Tables (DHTs) algorithms as in Chord [Stoica 01] or Pastry [Rowstron 01] are used to define the positioning and search content. There is little control over resource location, since it is automatically determined by the algorithms that keep the overlay structure. Resources might get replicated to unknown (to the user) peers.
  - Non structured: the network overlay is created in a non-deterministic way (ad-hoc) as the nodes and content is added. The resources remain at is owner nodes, or can be replicated to well known nodes.
- Topology

- Decentralized: All the nodes play the same role, with the same responsibilities and there is no centralized coordination on their activities.
- Hybrid: There are central servers that facilitate the interaction between peers, maintaining directories of meta-data about the peers or resources. However, the interactions are direct between peers.
- Partially centralized: Some nodes assume more responsibilities than others, playing the role of super nodes substituting the centralized meta-data servers. These can change over time.

Peer-to-peer architectures provide a scalable platform for applications, were the peers can both provide resources (or services) or consume other's resources (or services). The characteristics of the architecture influence the success of the application as searching algorithms in decentralized non structured systems do not grant the success (e.g. Gnutella) or have high latency, with the response not timely arriving. In dynamic environments with a huge number of nodes (like the Web), the algorithms for structured architectures have trouble stabilizing the hash tables, while in hybrid architectures, the centralized servers constitute bottlenecks. Partially centralized architectures must provide a set of super nodes that are stable enough in the network to grant the coordination of the remaining, or the system might be compromised.

3. The Browserver

This dissertation advocates that the user's entry-level tool for Web access should be a Browserver, which includes a browser (*B*) and a server (*S*)), as represented in Figure 3.1, hosted by his 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.

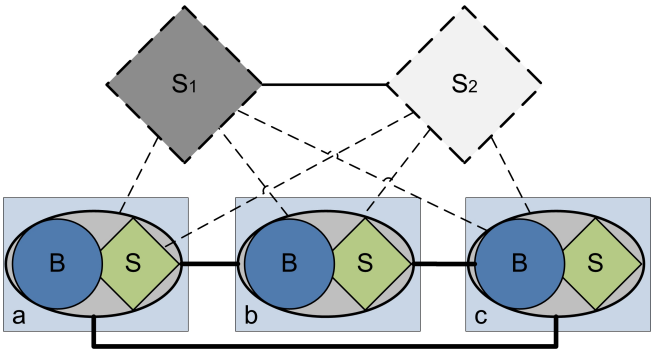

Figure 3.1: The Browserver, the union of a universal client and a universal server on P2P interactions.

By giving each Web user a server capable of executing web services, his capabilities can be modelled and executed as real web services, invocable by other entities or services, enabling of a user to interact with a business process given his own capabilities. Traditional solutions put the responsibility of representing the user as web services on the server side, building a UI that he can access through the browser using a Uniform Resource Locator (URL) [Berners-Lee 94a]. The capabilities represented are the ones determined by the analysts and programmers who define what a user can do in a business process.

3.1 Key Aspects of the Browserver

Although each user is able to create content and resources [O'Reilly 07] that others can use, he is still positioned in the edge, and not in the center of the Web. Security issues limit browser's connections to be made only with the originating server of the Web page that the user is navigating, making impossible to build applications for direct collaboration through the browser. Each

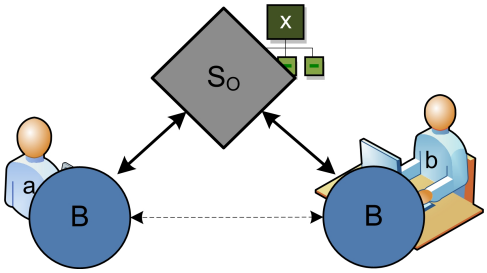

Figure 3.2: Client-server model. There is always an intermediary offering services to each entity.

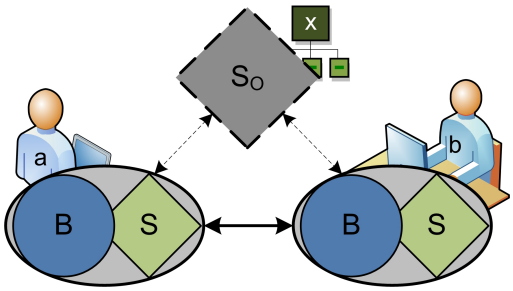

Figure 3.3: Peer-to-peer model. Entities can interact directly, consuming each other's services.

user acts as the ultimate consumer of services made available by other users or organizations on remote servers, that act as interaction intermediaries and have full access to information shared between peers even if private. The Browserver sets P2P (Figure 3.3), as its paradigm for interactions, instead of the classic client-server model of the Web (Figure 3.2).

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:

- A generic application platform. It is not targeted at any specific use, but it is suitable for the development of almost any web application (chat, e-mail, web sites, file transfer,...).
- A generic user interface for web services. It is both a user interface for executing services on user's demand, and a user interface for service's interaction with the user.
- Public and private services. Public services can be published an searched in the web, while the private remain for its owner 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 (their description documents) and searching for service providers, with the Browserver enabling the operations to the user.
- Relationship parity. Each Browserver is considered a peer with the same capabilities (as a provider and a consumer) and not restricted to one role of client or server.
- Fully asynchronous message communication which is granted by applicational (and informational) blind intermediaries (services Gateways, as described in the next chapter) on the

absence of the receiver.

- Direct peer-to-peer interactions with messages being pushed from the sender to the receiver. In other words, 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. Security mechanisms, such as encryption can add extra insurance to the privacy issue.
- 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 and how their presence is managed.
- Suitable for Web standards (as in the implementation described in this document).

3.2 The Browserver as a Personal Tool

Empowered with a personal Browserver, every user becomes an active participant of the web, creating value not only by using third-party provided services, but contributing for the growing Internet of Services with his services provided by his own.

Distributed user applications can be built knowing that that not only every user as a generic client enabling him to interact with them, but also that every user is capable of providing services and resources for the application. This enables new business models involving the end costumer. Individuals can fully operate their own businesses in the Web through the Browserver, eventually creating a wide network of collaborative personal connections, composing each ones abilities for the delivery of new electronic and physical products and services.

The Browserver enables direct collaboration and workflows with other entities, by consuming the services they have to offer and providing services for others to consume (Figure 3.4). Extending user's provided services equals improving is capabilities in the web. In the lower limit, besides normal web navigation, the Browserver enables others to interact with him through its user interface web services. On the other end, from simply receiving messages to complex document reviews, work scheduling or collaborative design of an architecture, the Browserver itself does not restrict the applicational limits (only possibly the underlying technologies).

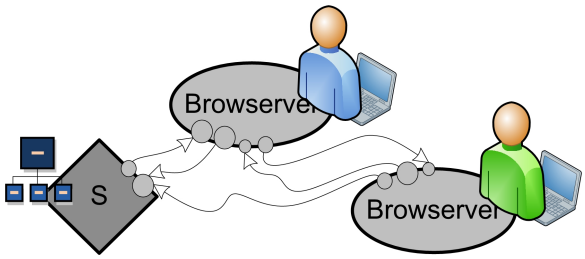

Figure 3.4: Service consuming and provisioning for individuals.

3.3 The Browserver as an Enterprise Tool

Inter-organizational business processes involve workflows of materials, information and knowledge between two or more organizations, possibly in a collaborative way and with common or distinct goals. The more efficient a business process is, greater is the competitive advantage in the market. Information systems enhance businesses by automating the workflows of information and improving the business processes of the organizations.

Small enterprises often do not have the resources to introduce the heavyweight and expensive information systems available. Therefore, the email continues to play a very important role in business relationships. Even bigger organizations still rely on the email for communication with some of their partners, and with most (or even all) of their customers.

The Browserver consists of lightweight solution that can offer even the smallest company the means to interoperate with their partners and improve costumer intimacy. Considering a widespread usage of the Browserver at the same level the browser is nowadays, with the architectural solutions described in this dissertation, an enterprise can easily integrate a supply chain even if their infrastructure boils down to a single laptop intermittently connected to the Internet.

Some scenarios involving business to business and business to client relationships, are described in this document, along with descriptions on how the Browserver suits those relationships.

3.4 The Browserver as a Workflow Platform

The Browserver constitutes an ideal platform for workflows involving electronic and human services. Figure 3.5 presents a BPMN [White 04] of simplified workflows for a credit loan application from a client (b) and for credit loan approval from a company's credit manager (a).

Using the Browserver as the platform for the execution of these workflows a the solution features among others:

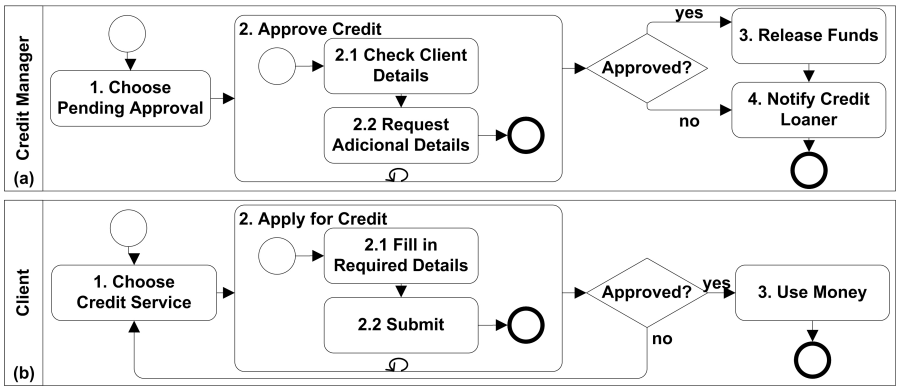

Figure 3.5: BPMN-modeled workflows for (a) credit loan application and (b) credit loan approval.

- The credit manager is warned of existence of new pending approvals assigned to him with a pop-up alert or a blinking service portlet in the Web portal, instead of having to constantly be checking for new tasks in a specialized workflow software tool or be warned by email;
- The assignment of a credit approval is done by balancing the workload of available managers and automatic conciliation of agendas between them. Browserver services work in the background of each manager's Browserver to achieve consensus in work assignment through well user-defined policies;
- The client does not have to fill in an exhaustive form to apply for credit. The process goes interactively. After the initial submission and approval, the system requests new data, according to the profile defined for the user and the feedback given by the manager. The requests are sent to the user using service interactions and compositions that automatically build the UI needed;
- The interactions are made directly between the client and the manager or between the client and a specialized company server for credit applications that forwards the requests to the managers;
- State is maintained in both sides, enabling complex business logics with temporally sparse activity from all involved entities. The Browserver provides a universal UI for services, allowing the development of any application.

A complete workflow and communication platform is designed for and executed in the same environment, without the need for external tools. Structuring information is possible and personal data mining services can arise, enabling any person to have personal Decision Support Systems.

4. Architecture

The Browserver includes both a local application standing at each peer's computer and the Web on which it operates. The next section introduces the service model of the Browserver while sections 4.2 and 4.3 present the application and the web of services where it operates, respectively and section 4.4 describes the interactions that the Browserver enables.

4.1 Services Model

Each user equipped with a Browserver is able to provide services to others and consume other's services in P2P interactions paradigm. The user can publish a service in the web (becoming a public service) so that it can be and discovered and consumed directly by other peer, who can be either another Browserver or a server from some organization. Either private or public, conceptually as in Figure 4.1, a service can be:

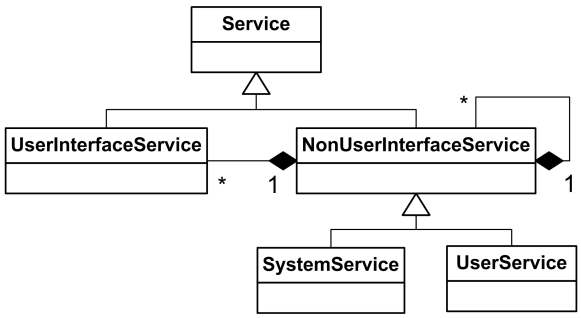

Figure 4.1: Browserver Conceptual Service Model.

- User Interface services. These enable the interaction with the user. Given that the action of the user over a generated interface for the interaction might be immediate or temporally sparse, this type of services only operate under the fully asynchronous communication mode.

- Non-interface services. These include system services and user deployed services. The latter, are the ones that represent the user's specific capabilities or extend the means to interact with other (external or internal) services. Non interface services can implement any interaction pattern enabled by the underlying technology, that way extending the communication modes with the user. Using the interface services, these can act as pre-defined interfaces, enabling complex sequential or parallel interactions with the user. The mainte-

nance of the state is responsibility of each service.

A service can be consumed by the user himself, or by another entity or service. If it is public, a service directory holds its description, which can be searched and then used for service consumption. If private, it can also be consumed by other entities as long as the user provides them the information necessary for it (e.g. the service description document). Figure 4.2 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. The consumption of the services occurs in a P2P basis, 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.

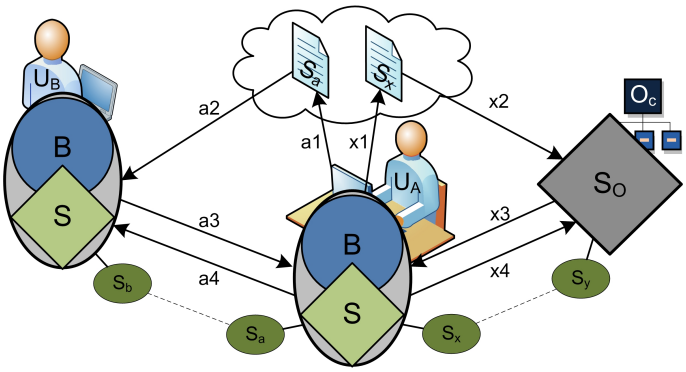

Figure 4.2: The user as a web services provider in interaction with another Browserver and a server.

Figure 4.3 shows part of a business process where the participants are an organization (through one of its servers) and a Browserver user, who is sequentially requested to execute service Sa3 (composed by S_{a1} and S_{a2}) and service S_{a1}.

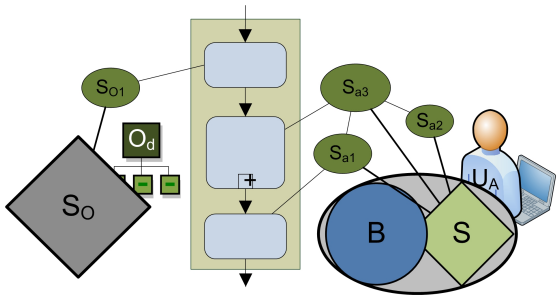

Figure 4.3: The user as a web services provider in a long-running business process with a server.

A Browserver might not be always available, and might be in that state for a long time. Also, its user can start a task, stop, continue, restart or cancel it at any time, so an answer can't be expected to be immediate or even assured. Non interface services can be consumed in any existing interaction pattern for web services, provided that they are developed to work in that

patterns and that these specificities are taken into account. To overcome these problems, the Browserver defines a fully asynchronous interaction mode, which includes services Gateways for message routing between peers. The messages include routing policies, defining which must be the behaviour of intermediate nodes (e.g. gateways) for each message and a reply endpoint, which is used for contextually related replies that can occur any time in the future. Details on this interaction mode are described in the next sections.

4.2 The Application

The Browserver application unites a generic web browser and a server capable of executing web services. The possible architectural solutions for this union include (a) developing a new web browser with an fully integrated server, (b) develop a new web browser connected to an application running on the server and (c) developing a server application to which any existing browser can connect. The architecture here described aligns with option (c), which enables the use of any browser with any compliant server, while allowing normal web navigation. Another advantage is the possibility of physical separation of both components. The server can be remotely located while the browser is, for instance, at the user's less computationally capable mobile device although, in this paper we consider that both are located at the same machine.

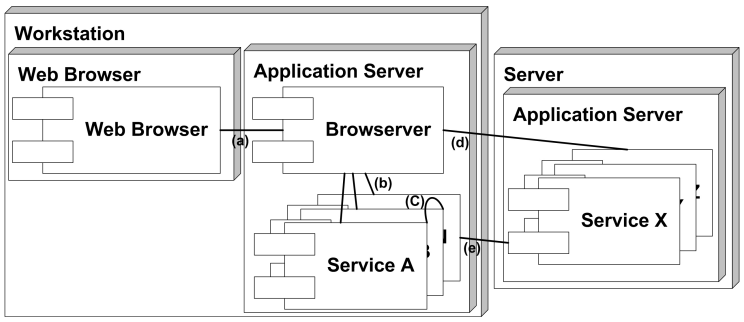

Figure 4.4: Architectural deployment view of the Browserver.

As shown in Figure 4.4, the Browserver consists on a locally deployed web application 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.

The Browserver application is divided in two main parts each containing two main components, as in Figures 4.5 and 4.6. The *Browser* (*B*) part divides into the *Browser Management* and the User Interface Management, while the *Server* part (*S*) contains the Services Management and the *Network Management*. In the context of this solution, the development leads to a single application deployed and running on an application server (as further described in the next chapter).

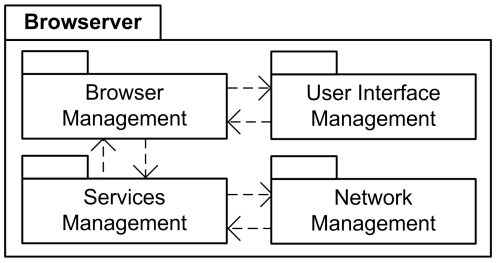

Figure 4.5: Main components of the Browserver application.

Figure 4.5 simplifies the main components of the system and their relationships, being each component's main responsibilities here summarized:

- *Browser Management*: Manage the communication with the browser, updating the displayed interface. Receives UI requests from the Services Management and sends back related user replies.
- *User Interface Management*: Creates, destroys and manages each active UI requested by the Browser Management. Converts the internal representation of the UIs to a browser-displayable representation using XSLTs or specific converter objects. The components responsible for this are the *BrowserManager*, the *Container* and the *ContainerUnit*s.
- *Services Management*: Contains and manages the services that are available for external (local or remote) consumption 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 4.6 shows a simplified logical view of the Browserver. The architecture is here covered in higher depth:

- The *BrowserManager*, coordinates the creation of *Containers* and *ContainerUnits*, and is responsible for sending the full *Container* UI for the specific browser that requires it through the *Proxy*, as well as creating new UI units from *UIData* sent by the *ServiceManager*, using the *UnitBuilder*.
- A *UnitBuilder* takes the XML definition of an UI and builds a *ContainerUnit* representing that UI. The *BrowserManager* can then add it to a *Container*.
- There can be one or more *Containers* available (the *Portal* is the base reference for this architecture) and each holds multiple *ContainerUnits*, *Portlets* and *ControlPanel* realizations of these. These elements produce code understandable by the browser like HTML and

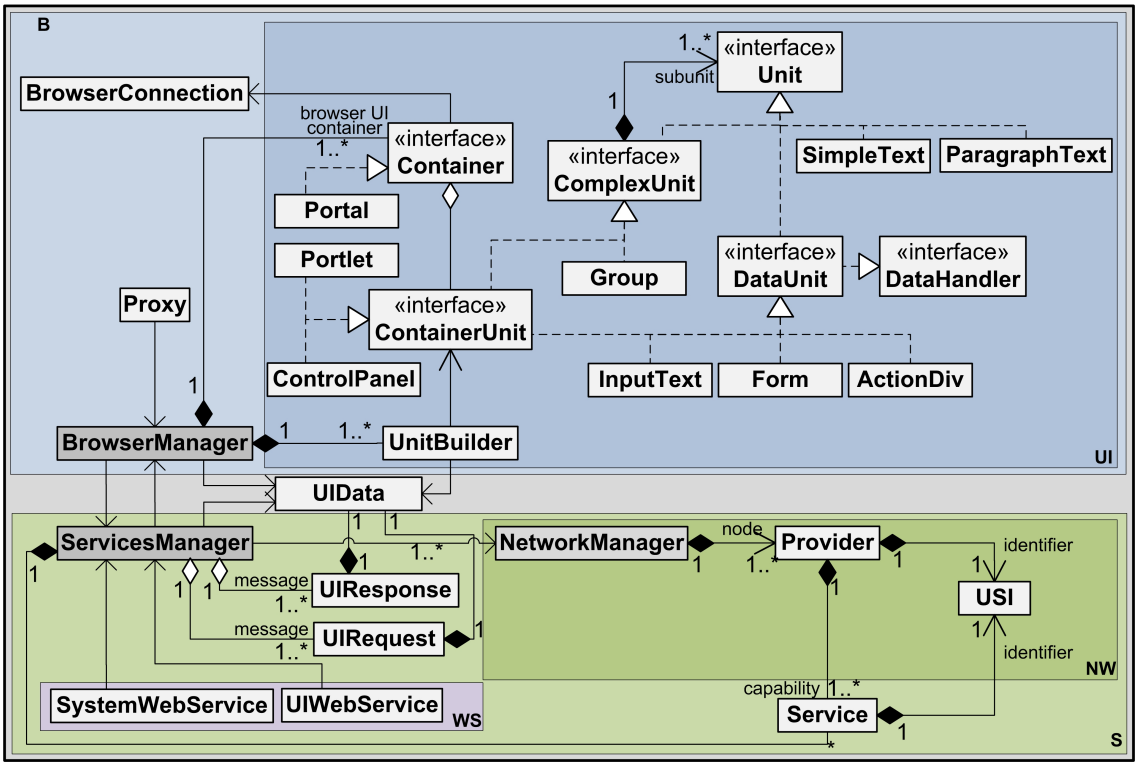

Figure 4.6: Simplified logical view of the Browserver.

Javascript. The *Container* also updates the UI at the browser through the *BrowserConnector* whenever it changes internally.

- The *BrowserConnector* component represents the link with the browser that enables bi-directional communication allowing the update of the browser whenever it is needed. In Chapter 5 Direct Web Remoting (DWR) [Marginian 09] is used to implement this component.

- A *DataHandler* has the ability to handle user input from the browser. Its corresponding *ContainerUnit* handles the data, sending it to the *BrowserManager*, who then forwards it to the *ServiceManager*.

- A Interface Unit can be:

  – A *SimpleUnit*, which cannot hold any other units inside (e.g. a *SimpleText* is used to present text without any special format).

  – A *ComplexUnit*, which can hold other units (e.g. in HTML, a `<div>` element plays this role).

  – A *DataUnit*, which is a *ComplexUnit* and *DataHandler* that collects data from the user (e.g. a `<form>` element in HTML corresponds to a *DataUnit*).

  – A *ContainerUnit*, which is a *DataUnit* that holds the whole UI for a service and handles

input data from the browser, redirecting it to the corresponding service at the Server part.

Different *Container*, *ContainerUnits* and/or *UnitBuilders* implementations enable different graphical frontends. For a mobile device, simply extending an existing *Container* into a new one that converts the output using XSLT is a feasible solution.

- The *ServicesManager* is responsible to manage incoming requests and outgoing responses for UIs and system operations, has well as the locally deployed and the remotely known web services (each belonging to one *Provider*).
- The *Network* consists of at least one *Provider* (the local host) and all the known remote providers that can have any number of associated *Service*s.
- The *NetworkManager* manage the known providers and offers means to remotely register local services and search for published services. A remote service directory is used to publicize the services of the Browserver.
- Each *Provider* has one or more associated *Service*s and all have one unique USI.

4.2.1 Base Services and User Deployed Services

The base services provided by the Browserver, and managed by the *ServicesManager* component are:

- *UIWebService*: A web service enabling the creation of UIs to interact with the user. Specific UI services can be developed using this generic service. For instance, a *requestAge()* operation of a non user interface service can make use of the *requestUI(ui)* with an ui definition containing a form with a simple text and field for the user to fill in and submit. Its only operation is:
  - *requestUI(ui)*: receives a UI definition that will be presented to the user. The execution of this operation as no immediate result. The user 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. Its main operations are related to deployment of services, which can be located locally or remotely. Here, deployment means registering the service as being a service provided by this Browserver and not the deployment in the server (which is done through server-provided means). The *uri* parameter of the operations is the web service definition document's (e.g. WSDL) URL.

- *deploy(uri)*: deploys a new service in the Browserver, if it already exists nothing is done.
- *redeploy(uri)*: deploys a service in the Browserver even if it already exists.
- *undeploy(uri)*: undeploys a service from the Browserver.

Apart from these two web services, the user can deploy new services that can make use of the previous to interact with him, as well services that execute always in the background, with no user interaction (as with a normal web services server). The interactions that happen for UI requests from both a local service *X* and a remote service *Z* to the Browserver are shown in Figure 4.7. 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)*.

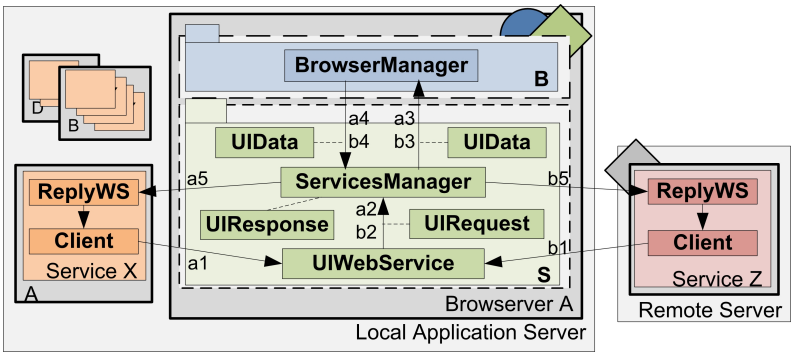

Figure 4.7: Interactions on UI request and reply.

4.2.2 Interface Generation and User Interaction

The generation of interfaces for services occur in two forms:

a. Generation of a user interface for the execution of a service on user demand.

b. Generation of a user interface for a request made to the UI services.

The first aims the case where the user wants to execute one of his own services or other remote service. Approaches to the creation of UIs in such conditions have been described in [Song 07, Kassoff 03]. Taking the service description (e.g. WSDL) as a basis, an HTML page with the necessary parameters of each service operation is created and presented to the user, who can fill and submit them, executing the service. The response is parsed and a UI is presented to the

user. The second case enables an entity to interact with the user on their own initiative, sending an UI definition that can contain complex elements such as forms for user data submission or simple informative text.

To request UIs to the Browserver, an entity consumes its *UIWebService*, sending an UI definition. A simple XML interface definition for a name and age request is presented in listing 4.1.

Listing 4.1: 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 *BrowserManager* creates (or re-uses) an appropriate *ContainerUnit* for the *Container* that targets the browser and further makes use of an appropriate *UnitBuilder* to parse the UI definition on the request. The *UnitBuilder* transforms the definition to an internal uniquely identified UI representation user interface *Unit*. This unit is then added to the *ContainerUnit* that is further added to the *Container*. The default *Container* (as in the implementation described in Chapter 5) of the Browserver is a *Portal* which holds *Portlets* as the *ContainerUnit*s 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 adapt to a different interpreter or a lower-end device such as a PDA or a mobile phone, the generated interface might either be transformed using XSLTs or new pairs of *Container*s and *Unit*s can be developed to generate specific UIs. Figure 4.8 summarizes the described interface generation for the request in listing 4.1.

When the user submits a form, the data is gathered and sent to the *Container* through the *BrowserConnector*. The *Container* forwards the data to the corresponding *ContainerUnit*, that parses and filters the data as needed. The resulting data is sent to the *BrowserManager* who forwards it to the *ServiceManager* to be sent as a data response to the corresponding requester. For the request in listing 4.1, the an example response data is represented in listing 4.2.

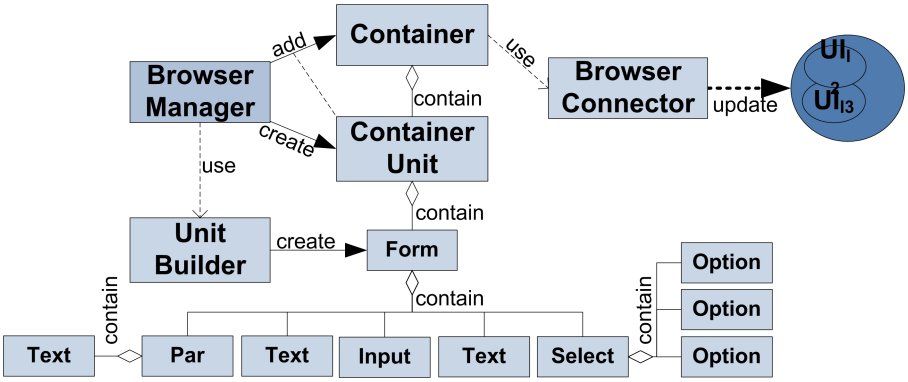

Figure 4.8: User Interface Generation.

Listing 4.2: UI data response sample

```
<data>
  <name>informationForm</name>
  <input>
    <name>name</name>
    <value>John</value>
  </input>
  <input>
    <name>age</name>
    <value>20-40</value>
  </input>
</data>
```

To contextualize UI requests, and reuse or replace an existing *Unit*, each UI request as an associated *relatesTo* unique identifier (UID). In Figure 4.9, the first message *(2)* sent by service *X* to the Browser requesting an interface has 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)* is 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 can be used for consecutive interactions with the user, keeping him in the same execution context.

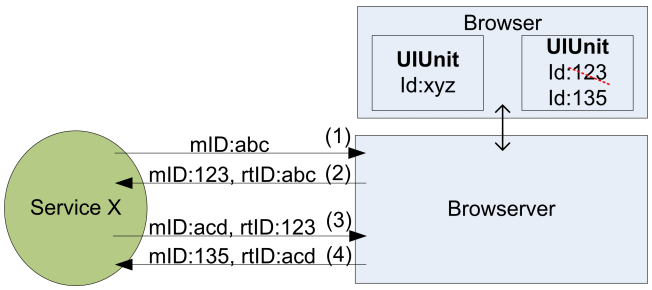

Figure 4.9: Interactions on UI request and reply.

4.3 The Web of Services

To enable the interaction between Browservers, it is necessary to know the location of its services but the location of the Browserver is highly dynamic, due to the topology of the Internet infrastructure (with dynamic IP assignment) and user mobility. Beyond location, the availability of the each Browserver is not granted at each time, just as its user. The Browserver web provides means to locate a service, to enable asynchronous communication and to allow the availability of a service even when its owner's Browserver is not available.

Distributed web service directories are described in [Du 06], a distributed directory based in the UDDI [Bellwood 04], and [Schumacher 07], a federated directory system for semantic web services based on OWL-S descriptions. The Universal Service Identifier enables service location and the services Gateway plays an important role in asynchronous communication for the Browserver.

4.3.1 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 [Berners-Lee 05], with the syntax:

$$usi : [Provider][/Service][/Operation] \tag{4.1}$$

$$[Provider] = [name]@[subdomain].[domain] \tag{4.2}$$

The following are examples of Browserver USIs:

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

The USI system (USIS), enables the assignment of identifiers to each service. Each public service must have its own USI and be registered at the USIS system, that offers *register*, *replace* and *unregister* operations through a web service. The system also offers web service query operations for service information, namely the location of the web service description document given an USI. 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 change that information in the USIS.

For the reference Browserver implementation previously described, the DUSIS system relates USIs to WSDLs locations, as exemplified in the following table from a node:

Table 4.1: 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

In a distributed USIS architecture (DUSIS), if the USI is not present in the primary node's tables, the query is forwarded to another node and results are cached. The ultimate USIS node is the Browserver to which the service belongs. The DUSIS plays a similar role as that of the DNS, and can be architectured just as similarly. The DNS hierarchical topology has given proofs of scalability and good performance and proposals [Cox 02, Ramasubramanian 04] based on Distributed Hash Tables [Stoica 01] while underperforming in normal operation (with normal node failures) offer better resilience to orchestrated attacks [Pappas 06].

4.3.2 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.

The Browserver services Gateway is based on [Venkatapathy 02], but the aim of becoming not only a application that can be running at an enterprise's local infrastructure for strict usage with their services, but a global, generalized and distributed node for communication support.

Services Gateways act as proxys to services, enabling fully asynchronous communication between entities. To enable such interaction mode, the service description must be published to a Gateway thus becoming a proxied service, however the service itself remains at its provider. When publishing a service description document to a Gateway, it becomes a virtual service provider for that service. The document is parsed and a new one is created wit all the endpoints changed to a Gateway endpoint that will handle the incoming requests. Just publishing the service isn't 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 4.1 shows a service which description location is set to a Gateway.

The Gateway does not inspect the body content of any message, strictly limiting its action to only interpret the necessary routing information. The implementation here described uses Java HTTP Servlet to concretize the Gateway, 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). The Gateway processes only SOAP 1.2 [Nielsen 03] headers that are marked with a *next* role. Listing 4.3 present a SOAP message with addressing headers. The headers processing takes in consideration:

- a specific *canQueue* parameter in *To* element's referenceParameters. This parameter indicates whether a message can be queued if its final recipient is unavailable. If not and if the parameter is set to false, a fault message is sent to the *faultTo* endpoint (if it exists and is its role is *next*). If the parameter is absent, it is assumed as true.
- if either the *ReplyTo* or *FaultTo* elements are not to be processed (the role is not *next*) then it is assumed that a response or fault is not sent through the same Gateway, otherwise, the WSDL for the reply is retrieved published in the gateway node to act as a proxy to a future reply or fault message and the forwarded message will have the Gateway as the *ReplyTo* endpoint.
- parameters *wsdlLocation* on *ReplyTo* and *FaultTo* endpoints. If absent, the address is used to resolve the WSDL location using the USIS, if not, the WSDL is retrieved from this location.
- parameter *deadline* in *ReplyTo* element, which indicates what is the date and time limit for a reply. A reply sent after this date is not guaranteed to be accepted or processed. This parameter is replicated in forwarded messages.

Listing 4.3: Addressing headers

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
        xmlns:wsa="http://www.w3.org/2005/08/addressing"
         xmlns:bsp="http://ws.browserver.com/parameters">
  <env:Header>
    <wsa:To env:role="http://www.w3.org/2003/05/soap-envelope/role/next">
      <wsa:Address>John@xpto.com/UIWebServiceService/requestUi</Address>
      <wsa:ReferenceParameters>
        <bsp:canQueue>true</bsp:canQueue>
      </wsa:ReferenceParameters>
    </wsa:To>
    <wsa:From env:role="http://www.w3.org/2003/05/soap-envelope/role/next">
      <wsaAddress>Mary@ztx.com</Address>
    </wsa:From>
```

```
    <wsa:ReplyTo
      env:role="http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver">
      <wsa:Address>John@xpto.com/MailWebServiceService/handleResponse</Address>
      <wsa:ReferenceParameters>
        <bsp:wsdlLocation>
          http://a.a.a.a:b/BrowserverMail/MailWebServiceService?wsdl
        </bsp:wsdlLocation>
        <bsp:deadline>2010-08-30T23:59:59Z</bsp:deadline>
      </wsa:ReferenceParameters>
    </wsa:ReplyTo>
    <wsa:FaultTo env:role="http://www.w3.org/2003/05/soap-envelope/role/next">
      <wsa:Address>John@xpto.com/MailWebServiceService/handleFault</Address>
      <wsa:ReferenceParameters>
        <bsp:wsdlLocation>
          http://a.a.a.a:b/BrowserverMail/MailWebServiceService?wsdl
        </bsp:wsdlLocation>
      </wsa:ReferenceParameters>
    </wsa:FaultTo>
  </env:Header>
  <env:Body> [omitted. unprocessed] </env:Body>
</env:Envelope>
```

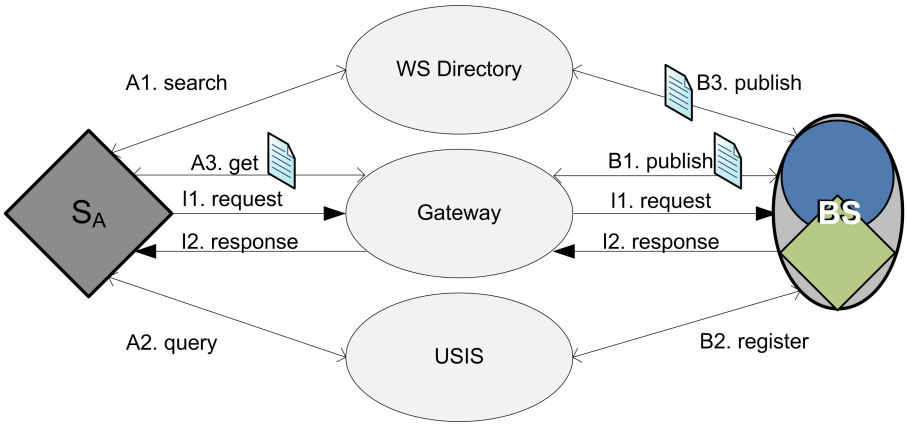

Figure 4.10: Service consumption through a Gateway.

Figure 4.10 presents the consumption of a service that is proxied in 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 USI is used to query (*A2*) the USIS for the service description document location, which is 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, with routing policies defined by SOAP header elements.

4.4 Interactions

Being a web services server, the Browserver enables any message exchange patterns (MEP) of the technology supported by the server, with any entity capable of consuming web services, under normal operational conditions for that technologies. If a Browserver operates only under such conditions, those message exchange patterns are supported by it.

Most interactions involving web services are not completely asynchronous. Although some platforms enable asynchronous service consumption (e.g. Java JAX-WS with Java Futures), this only enables the client to not to have to immediately wait for the result and retrieve it or being reactivated at its arrival. At the service provider side, the server creates the means for the reply given the description of the service and the protocol information, assuming that some computation is to be done by the service and the result is eventually sent. Browserver interactions must assume that the recipient might be unavailable, and being user-aimed, it must assume that the computation (the user action) can be done now, some time in the future, or never. Using W3C web services [Weerawarana 05], as in the implementation described in this paper, normal operational conditions do not coadunate with the unstable availability, and unpredictable reply time of each Browserver peer.

4.4.1 Fully Asynchronous Interaction

The fully asynchronous interaction mode of the Browserver, takes the unstable operating conditions in consideration and defines another interaction mode for Browserver services. Using only one-way messages, each request defines a *replyTo* endpoint where the response (if one) is to be sent, being no longer necessary that neither of the sides 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.

To further set the communication as fully asynchronous, enabling (virtual) service consumption even in the absence of its provider, the Gateway plays an important role. 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, ever UI service of the Browserver operates only under this asynchronous communication mode.

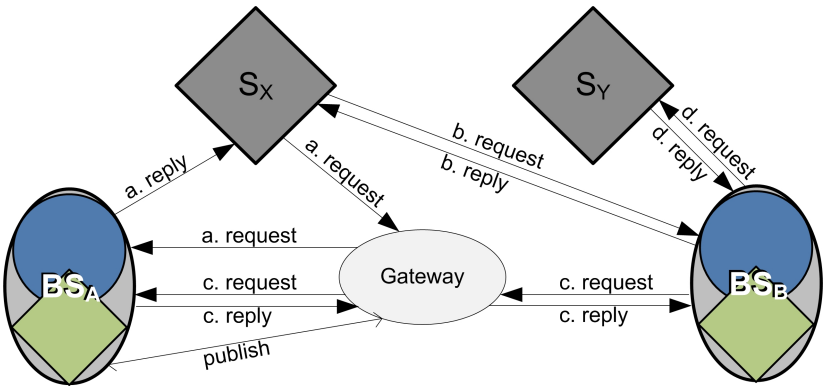

Figure 4.11: Interactions involving the Browserver.

In Figure 4.11 *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.

The full assynchrony of the Browserver also contemplates a deadline reply policy for the interaction, enabling a generic mode of defining a time limit for a data submission or a related reply by the user to which a request was made. Using the Simple Object Access Protocol (SOAP) [Weerawarana 05], as in the reference implementation of Chapter 5, this policy is set through a *deadline* parameter in the `<replyTo>` element of the message header.

4.4.2 Security

The Browserver enables direct interaction between two entities, minimizing the presence of intermediaries in the communication. Even if an interaction goes through any intermediate node, this node does not act at applicational level, restricting its action at communication level, thus being blind to the contents of the messages being exchanged. Although, as messages travel the Web, it is necessary to grant their security from third parties that can be *listening* for messages in some communication channel. Securing messages can be achieved through existing standard encryption and signature mechanisms.

Using SOAP as the message exchange protocol for web services, each message body can be encrypted for critical services. However, as Gateways need to interpret and change the headers, these must travel unencrypted if the interactions go through those. If the interaction pattern is other than the fully asynchronous described in this document, all the security mechanisms enabled by the underlying technology of the Browserver are possible. HTTPS can be the basic mechanism in this latter case.

5. Implementation

As a proof of concept, a working high-level prototype of the architecture presented in the previous chapter has been implemented, using existing tools and technologies. The most effort was put on implementing the application as a user interface for web services. As for the Web where the Browserver operates and to support the fully asynchronous message communication, the Gateway was the component where more time was invested. Since the service directory is not an architectural product neither an objective of this dissertation, it was not implemented.

The Browserver was implemented 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 as the graphical frontend for the UIs for the services, respectively. Glassfish enables deployment of new applications without restarting the server, which allows adding new web services developed as small web applications without restarting the Browserver. Firefox, Chrome, Internet Explorer and Opera have been tested and work with the solution.

5.1 The User Interface

5.1.1 Browser Connection and User Interface Management

To exchange data between the browser and the Browserver and update the UI displayed to the user three approaches were possible:

i. The Browserver acts as a passive web application and waits for user's requests for its interface through a URL in the browser

   Pro: Non intrusive application.

   Con: The actualization of the Browserver's UI is dependent from a explicit request from the user.

ii. The browser sets the local server as a web proxy through which it connects to the Internet.

   Pro: The Browserver can wrap the requested pages and it's UI in a new page response.

   Con: Intrusive, as the content of the pages the user is seeing are changed. The actualization of the UI is dependent from user's web navigation.

iii. Actively connect the Browserver with the browser through AJAX techniques [Crane 08].

Pro: The actualization of the UI is nearly real time on any event at the Browserver.

Con: The use of AJAX can be erratic or blocked on some browsers.

In any of the options, if there is any event or information needing user's attention and if no browser is pointing at the portal, the Browserver can request the operating system to open the default browser, pointing at the portal's URL. If such request is made and a browser is already pointing at that URL, the default action is to open a new page and not to refresh the existing one.

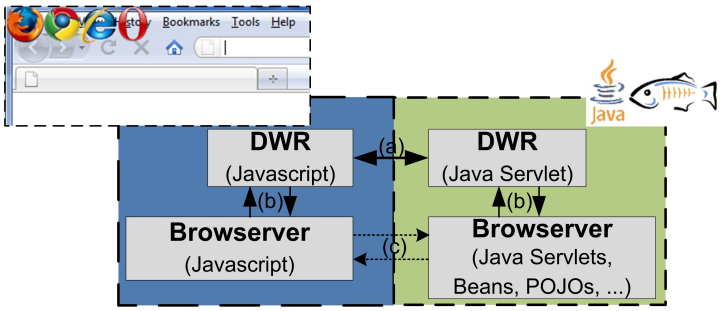

Figure 5.1: Connection between the browser and the server through DWR.

The last option was the choice that best fitted the Browserver. This way the user does not have to be concerned with checking if anything new as happened. The Browserver will update the UI with new events and requests that arrive for the user. The *BrowserConnector* component has been achieved through the Direct Web Remoting (DWR) [Marginian 09] library and JQuery [Resig 10] Javascript library at the browser side. DWR has been used for reverse AJAX communication between the browser and the Browserver application an JQuery to update the DOM structure of the visualized page and gather form data that is then submitted through DWR. As shown in Figure 5.1, 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)*.

As XML is used to define interfaces and the response to the submitted data, the JDOM[1] library has been used to parse incoming and create new XML documents, for its completeness alied with the ease of use. Since the default data format used in Javascript is JSON[2], the submitted data is sent to the Browserver as a JSON string that is further converted to a JDom object.

The implemented *Container* of the Browserver is the *Portal* which holds *Portlets* as the *ContainerUnit*s and builds an interface with a window system like metaphor, achieved through HTML, CSS and JQuery. Figure 5.2 presents the Browserver portal with some active portlets (*a,b,c*). The portlets can be dragged and dropped, minimized, maximized, restored and closed.

[1] JDOM: http://www.jdom.org/, last access on 2010-08-20
[2] JSON: http://www.json.org/, last access on 2010-08-20

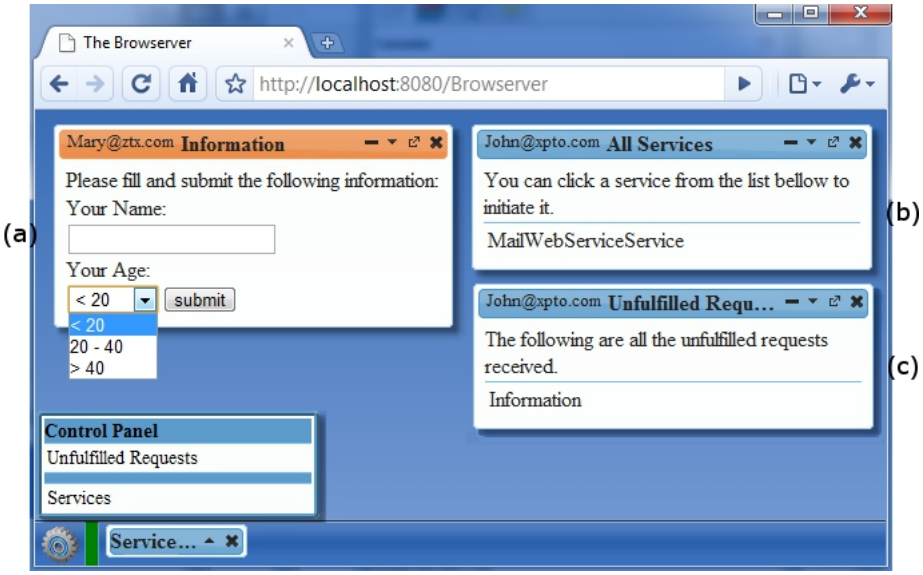

Figure 5.2: The Browserver Portal.

The control panel gives the user access to some useful tools. In this case, it 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 4.1 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.

5.1.2 User Interface Units

The user interfaces that are added to a *ContainerUnit* are result of parsing the user definition sent by the requester, to an internal representation and then to a frontend (the browser) representation. Although it is a user interface for web services, the main focus of this thesis was not the usability, neither how extense the user interface definition is. To simplify the prototype, a XML schema has been defined (see Appendix A for the full XSD). The definition holds the set of elements needed for this prototype, and can be extended or replaced in the future, provided as the a *UnitBuilder* is created or updated to comply with the definition.

Each user interface element has a corresponding internal *Unit* object, that represents it, and can output its individual definition to the final browser representation, which is HTML. The Browserver introduces interface elements related to its service consuming and provisioning characteristics. A `<knownProviders>` element inside a `<select>` element results in a selection element with all providers of a given service (as a required attribute of the `<knownProviders>` element) and `<date>`

element in an `<input>` element introduces a date picker user interface (used in the mail service described in section 5.2.1.

5.2 The Services

The basic deployment of the Browserver application includes the two base web services described in Chapter 4, the *UIWebService* and the *SystemWebService*. Figure 5.4 shows this situation, with a portlet where the two services are presented to the user.

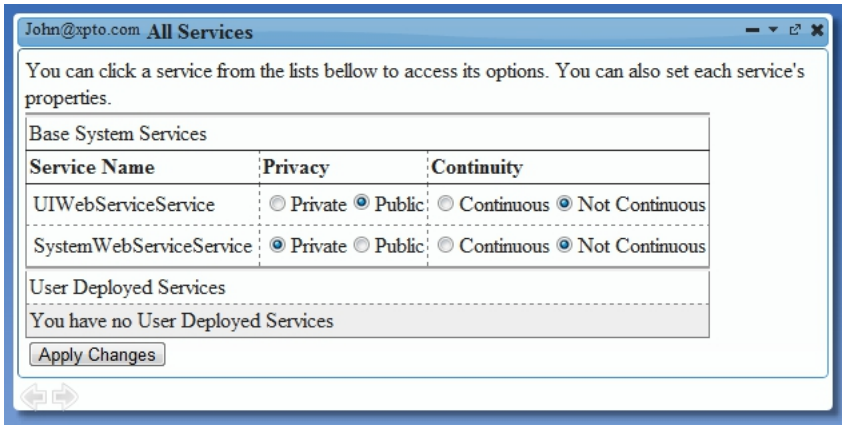

Figure 5.3: Browserver's base services, after the deployment of the Browserver web application.

Adding new services to the Browserver is a matter of developing new web applications and deploy them to the server, with a few details to observe: (a) to be managed by the user the *deploy()* operation of the Browserver's *SystemWebService* must be used with the location of the WSDL of the service; (b) an operation *execute()* defined in the WSDL is assumed as an operation for the first user interface interaction of the user; (c) the privacy of the service is defined by the user and only him can set it as public; (d) the continuity of the service is also decided by the user. This items must be taken in account when developing new services.

Figure 5.4 shows two new services as the result of the deployment of a new web application. The *UIMailWebService* corresponds to a service with only an *execute()* operation, acting as an entry point for the first interaction with the user, while the *MailWebService* is a business level service with operations for receiving new mail messages as well as defining the endpoints for asynchronous responses and faults. Figure 5.5 shows options for executing the *UIMailWebSer-vice* or view its operations.

As the Browserver is both an interface for services or entities to interact with the user, by request-ing user interfaces, and a user interface for executing service operations, when the user chooses to see the operations of a service he is lead to an interface that enables him to execute any of the

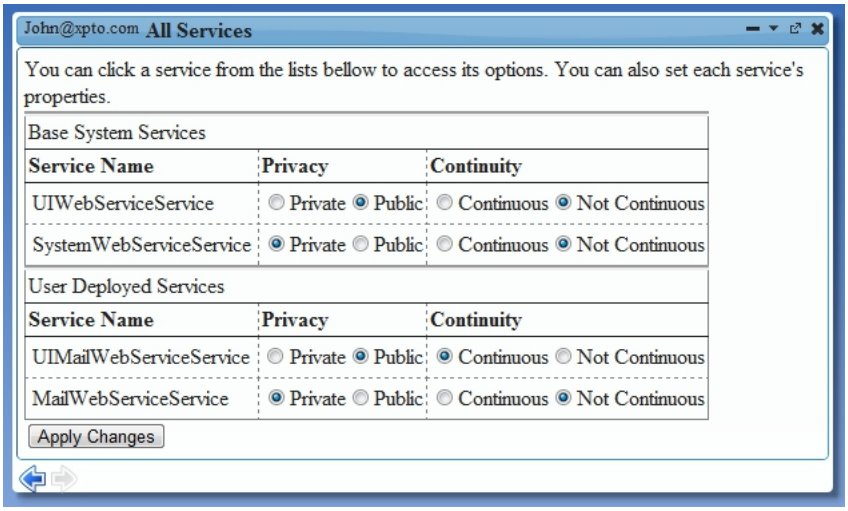

Figure 5.4: Browserver's base services, after the deployment of a new service *MailWebService*.

Figure 5.5: Service execution options.

operations (Figure 5.6). Having an internal representation of the WSDL of the service, building such interface is a matter of creating a form for each operation, with input fields for each of its parameters. In this prototype, complex objects (with complex XML schemas) as parameters are not addressed.

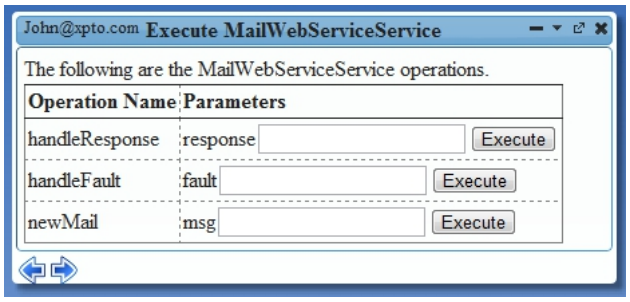

Figure 5.6: Operations of the *MailWebService*.

By default, a service is set as private and not continuous, and the user can change this properties. The table 5.1 presents the actions taken by the Browserver when changing this properties, considering the following actions:

- *publish(usi,url)* and *republish(usi,url)* as the publication of a service to the USIS and to the service directory, given its WSDL location (either at the real provider or the Gateway proxy

service), and *unpublish(usi)* the removal of the service from those systems.

- *proxy(usi,url)* as the method to proxy a service in a Gateway returning the proxyUrl for the new WSDL and *unproxy(usi)* the removal of that service form the Gateway.

Table 5.1: Actions on service state change

Previous State		New State		Actions
Private	Not Cont.	Public	Not Cont.	publish(usi,providerUrl);
Private	Not Cont.	Private	Cont.	proxy(usi,providerUrl);
Private	Not Cont.	Public	Cont.	proxy(usi,providerUrl); publish(usi,proxyUrl);
Private	Cont.	Public	Cont.	publish(usi,proxyUrl);
Private	Cont.	Private	Not Cont.	unproxy(usi);
Private	Cont.	Public	Not Cont.	unproxy(usi); publish(usi,providerUrl);
Public	Not Cont.	Private	Not Cont.	unpublish(usi);
Public	Not Cont.	Public	Cont.	proxy(usi,providerUrl); republish(usi,proxyUrl);
Public	Not Cont.	Private	Cont.	proxy(usi,providerUrl); unpublish(usi);
Public	Cont.	Private	Cont.	unpublish(usi);.
Public	Cont.	Public	Not Cont.	unproxy(usi); republish(usi,providerUrl)
Public	Cont.	Private	Not Cont.	unproxy(usi); unpublish(usi);

5.2.1 The Mail Service

An email system has been developed over the Browserver platform, implemented as a web application, empowering the user with the capacity to send and receive emails to and from others. The system consists of:

- A Business layer with the application logic of the email, with an inbox, an outbox and operations for sending and handling received messages.
- A Web Service (*MailWebService* with operations: *newMail(msg)*, to receive new email messages; *handleFault(fault)*, as an endpoint for faults, *handleResponse(response)* as an endpoint for data responses.
- A Web Service (*UIMailWebService*) with only an *execute()* operation, to start the interaction with the user.
- A Servlet initialized at deploy time, to open the browser pointing to the configuration pages.

- JSPs and Java Beans for the configuration page. The pages implemented serve to deploy the services to the Browserver.

After deployment, the Servlet initializes the application, which searches for a Browserver at the localhost. The browser is the requested to open the configuration page, where the user must choose where to deploy the email service (Figure 5.7). The deployment causes the Browserver application to retrieve the WSDLs for each of the services, and from that point, the user can start using those services through the Browserver user interface.

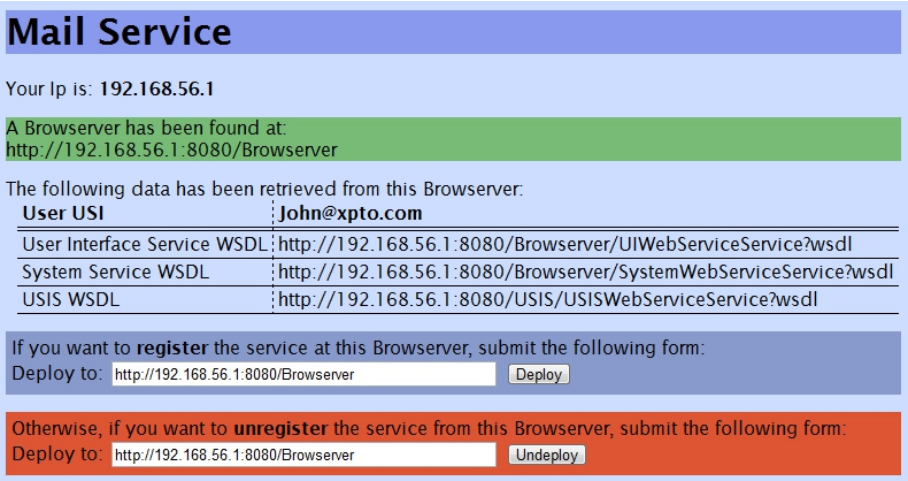

Figure 5.7: Deployment of the mail system.

Figure 5.8 presents the interactions that occur to send an email, considering that this is the first user interaction with the service. The user *A* requests *(1)* the execution of the mail service, through the *UIWebService execute()* operation. The *ServiceManager* consumes *(2)* the service which forwards *(3)* the request to the *Mail* business level object. An email interface *(4,5,6)* is then sent to the user, who fills the message and submits *(7)* it to be sent. The *ServiceManager* sends *(8)* the submitted data to the mail service, through the *handleResponse()* operation endpoint that delivers *(9)* it to the business level. The *Mail* consumes *(10)* the *newMail()*, therefore sending the message to the user *B*. The *Mail* at *B*'s side requests an user interface *(11, 12, 13)* with information of the newly arrived message, which is this way shown to its addressee.

Note that in *(10)*, the message might go through a Gateway, if the service is set as continuous by the user *B*. If not and the user is not available, the message will fail to be sent and the user *A* warned of its failure.

Figure 5.9 presents two portlets that are result for requests made by the mail application. The first presents the list of emails in inbox and in the outbox and a button to write a new email. Clicking in this button results in a new portlet with a user interface allowing the user to write and send a new message. The user is given the chance option to set a deadline for a related reply, which

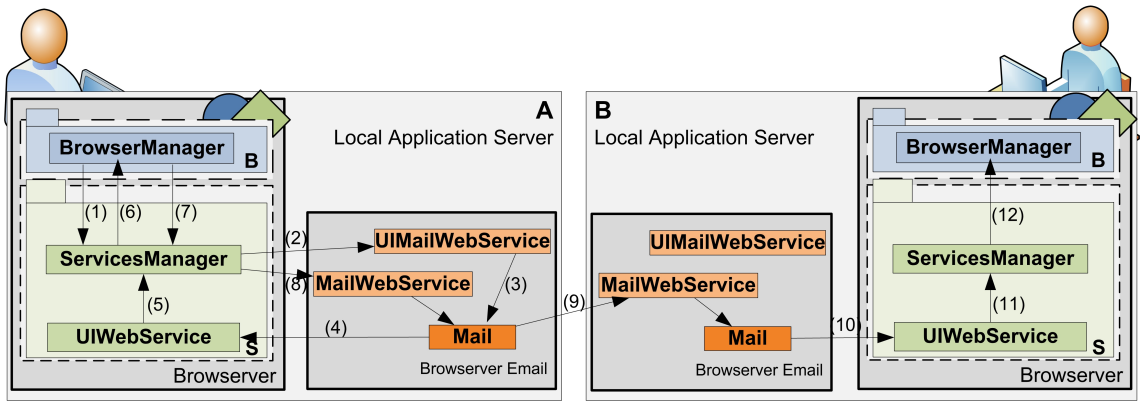

Figure 5.8: Interactions in the email system.

is sent as a `<replyTo>` parameter in the SOAP header. When viewing a received message, the user interface presents a reply option that leads to a new user interface for a reply, as in Figure 5.10. The reply message uses the `<relatesTo>` parameter in the SOAP headers to relate the messages.

5.3 The Gateway and the Universal Service Identifier System

As described in Chapter 4, the Gateway acts as a proxy and the USIS enables the Universal Service Identifier for web services. As a prototype, both systems were developed as single centralized nodes. Not their operation as distributed systems but rather their functionality as been the focus of the implementation.

The more relevant Gateway web application implementation aspects are described:

- A *GatewayWebService* enables publication operations for a proxied service offering the operations:
  - *proxy(providerUsi, wsdlUri)*, which is used to proxy a service given its location (*wsdlUri*) for the provider with *providerUsi* USI.
  - *reproxy(usi, wsdlUri)*, which tells the Gateway to change the WSDL with *usi* USI for a new one (*wsdlUri*).
  - *unproxy(usi, wsdlUri)*, that removes a service from the Gateway.
- The *GatewayServlet*, that handles incoming SOAP messages sent through HTTP with a URL pattern `/*`
- A *FileServlet* handles HTTP requests for files (the WSDL and XSD files) for requests in the pattern `/resources/*`

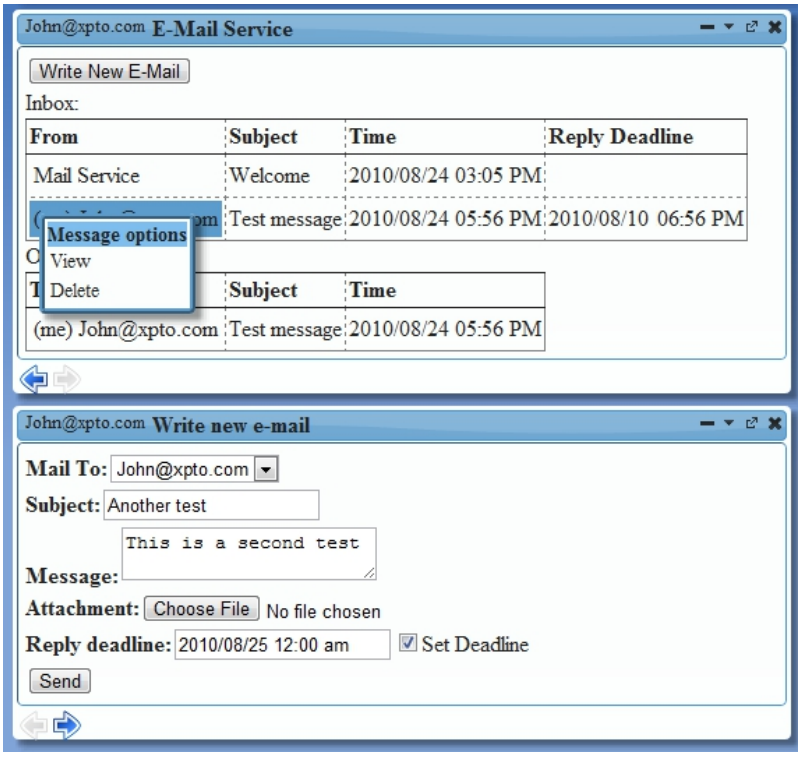

Figure 5.9: User interfaces of the mail application - message list and write a new message.

- Domain logic objects for retrieving and parsing WSDL files, support the *GatewayWebService* operations and routing incoming SOAP messages.

Upon receiving a request for proxying a service, the following actions are taken by the Gateway:

1. The WSDL is retrieved through an HTTP GET request and:
   (a) The retrieved WSDL is parsed to:
      - Determine the services and operations it contains. Each service and port is given an unique identifier.
      - Get the location of each external schema file.
   (b) A new WSDL is created and stored with an unique id, as a copy of the retrieved, with all the URLs set to a Gateway URL. Each service url is set as:
      - `[GatewayURL]/?id=[wsdlId]&s=[serviceId]`
      - Example: `http://123.456.78.9:8080/Gateway?id=a1d31s&s=1`
   (c) Each schema file is retrieved through an HTTP GET request, and stored with a unique identifier.
   (d) The new WSDL and all the schema files are given URLs in the form:
      - `[GatewayURL]/resources/[wsdlId]|[xsdId].wsdl`

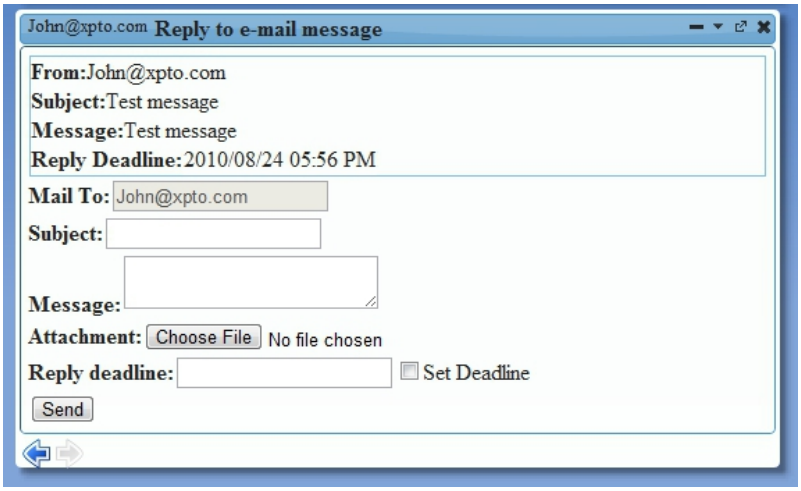

Figure 5.10: User interfaces of the mail application - reply to message.

- Example: `http://123.456.78.9:8080/Gateway/resources/a1d31s.wsdl`
- Example: `http://123.456.78.9:8080/Gateway/resources/a1d31s0.xsd`

2. The location of the WSDL for the proxied service is returned to the requester.

The WSDL retrieved from the Gateway, enables the service client to create the necessary dispatchers, while directing all the requests to the Gateway instead of the real provider, this way acting as a proxy. The handling of SOAP messages is made as described in section 4.3.2. Depending on the headers, a request received at the URL `http://123.456.78.9:8080/Gateway?id=a1d31s&s=1`, can be forwarded to the original service URL, queued (with periodic re-attempts) if the service is unavailable or a failure message sent to the requester if it is unavailable and the message can not be queued.

The implemented USIS consists of a single node providing the basic functionality for the system. Deployed as a web application, similarly to the Gateway, the USIS consists mainly of:

- A *USISWebService*, with operations:
  - register(providerUsi, wsdlUri), enabling the registration of services given the WSDL location.
  - replace(providerUsi, wsdlUri), for the replacement of an existing registration with a new WSDL location.
  - unregister(usi), to unregister a service.
  - getWsdlUriForService(usi), to get a the WSDL location for a specific service.
  - getAllProvidersForService(usi), to get the USIs of all providers of a given service. The USI parameter can correspond to the full form or just the service part.

- Domain logic objects for retrieving and parsing WSDL files, maintaining the service tables and support the *USISWebService* operations.

6. Application Scenarios

In this chapter, two business scenarios are presented, to show how the Browserver can be a valuable platform for the business processes that take place between the parties involved. The first scenario regards a B2C relationship, where a company interacts directly with the end customer that requests one of their services. The second scenario focus mainly on the supply chain management between three companies. The solutions, rather than extense designs are mainly high level descriptions over those designs.

6.1 Business to Client

Scenario 1: John bought a new car and wants to build a garage for it in his backyard. After a searching for a constructor, he asks the BuildFast company for a construction budgeting. For the budgeting, BuildFast needs information from John regarding the dimensions and materials to use in the garage. After gathering all the project information, the budget is compiled with collaboration of different area managers (architecture, carpentry, electricity,...). The final budget is approved by the top manager and sent to the costumer, who is given the possibility to accept or reject the offer within 7 days. A BPMN [White 04] workflow representing the set of interactions of this scenario is presented in Figure 6.1.

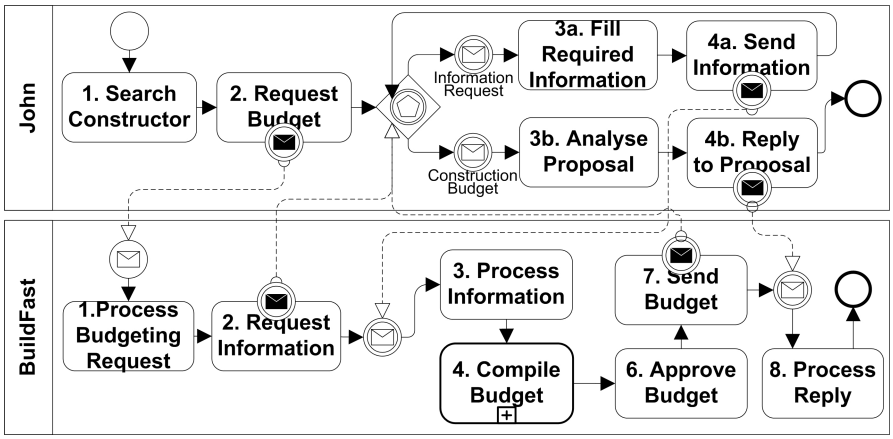

Figure 6.1: BPMN for construction budgeting - Scenario 1

When developing information systems to support interactions of this scenario, among others some questions arise. For each question, a Browserver-based solution is presented.

Question How can John find and request BuildFast budgeting service?

Solution BuildFast develops a generic budgeting web service, publishing it to a publicly available web services directory. With his Browserver, John searches the directory, finding all the companies offering the required service. Choosing BuildFast (and eventually others), John requires the construction budgeting by consuming the web service.

Question How is all the necessary project information required to John?

Solution BuildFast requires all the information at once or iteratively, consuming the services on John's Browserver, that enable them to request information on a asynchronous peer-to-peer fashion.

Question How does the information flow inside BuildFast?

Solution Each employee has his own Browserver, with deployed services that enable collaboration between them. A chat service enables conversational capabilities to the group and a dedicated collaborative budgeting service does the distributed information sharing between all participants.

Question How does BuildFast require a deadline for the acceptance of the offer?

Solution The message sent to to John containing the budget contains a form for the reply, a reply policy indicating the response deadline, as well as textual information about the meaning of the reply.

The following are some of the advantages of a Browserver solution over traditional solutions:

- No dedicated continuous server is required at any of the parties. The Browserver Web grants a communication channel even in the absence of one of the ends.
- Sharing information between a group of collaborating persons does not require a third-party application nor even an intermediate applicational server, it's peer-to-peer.
- If the organization needs additional information from the client, it can be provided using the same platform (and not by sending an email, or by a phone call) by making a contextually related request directly to the client's Browserver.

6.2 Supply Management

Scenario 2: SmartPrice Market (SPM) is a small local supermarket with 25 employees. The inventory list is kept in a dedicated local database, that is connected with the cash registers. Each product has a minimum stock limit, when this limit is reached its time to re-order. The

stock management is done by the SmartPrice manager, Mark, who decides and makes all the orders. The orders are made to a considerable number of suppliers, that by their turn, have their own suppliers and so on.

After reaching the lower limit of popular brand of low cost bicycles, the database triggers an alert. Mark compiles an order and sends it to Two Wheels Power (TWP), their bike supplier and manufacturer. Having a just-in-time inventory strategy, TWP needs to order parts to build the bikes. Among TWP's parts supplier there is Rubber Classics (RC), who receives their tires order. Having the tires in stock, RC ships them through DHL to TWP. After receiving all the needed parts and, TWP builds the the bikes and the order is shipped to SPM through DHL. During the processes, all the companies want to track the stage of each order. Figure 6.2 presents the workflows of this scenario.

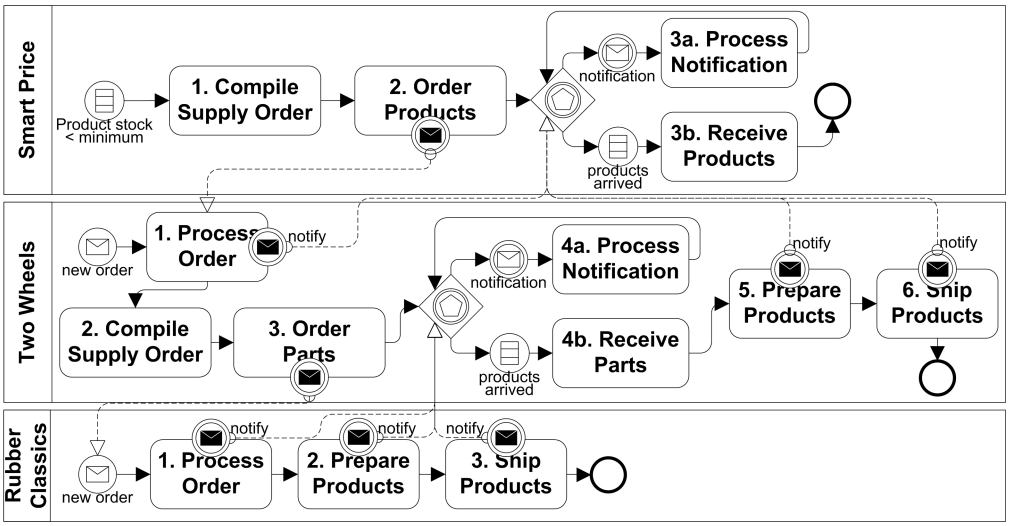

Figure 6.2: BPMN for supply management - Scenario 2

Using the Browserver, a system enabling the information flow on the the supply chain management of scenario 2 can be developed. Figure 6.3 presents a simplified architecture design for this scenario.

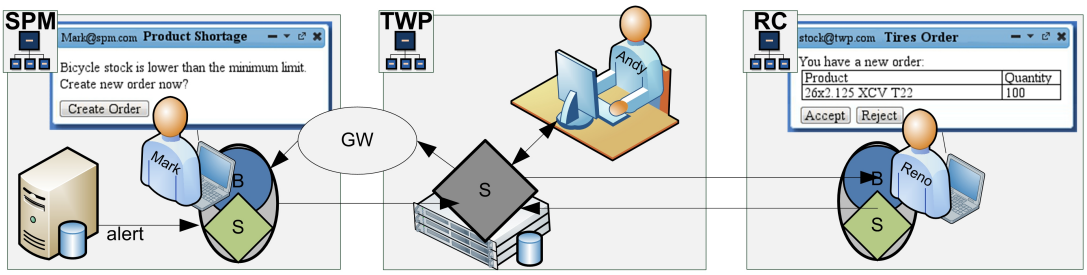

Figure 6.3: Simplified architectural view for Scenario 2.

- The infrastructure at SPM consists of a 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 services Gateway (GW) enables asynchronous communication with SPM.
- TWP has redundant systems with business applications specifically developed to suit their needs. A server is always available to external communication with their business partners. To interoperate with the Browserver, compliant services 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.

Apart from the transportation company and considering only the three represented organizations, the order process involve the following interactions:

1. When the minimum product level is reached, a database script triggers an alert.
2. Mark's Browserver contains a service (only internally available) that is used by the database to alert for product shortages. This service displays a UI with an alert for product shortages, giving him the possibility to create a new order.
3. A private order service enables Mark to fill request orders through the browser interface and dispatch them to the correct suppliers' order services. The order already has all the necessary products pre-filled, based on all alerts received and not yet attended.
4. Using the order service, Mark reviews and an sends the order to the TWP. The message indicates an *replyTo* endpoint, which location is set to a Gateway.
5. 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.
6. All the relevant notifications are sent to SPM through the notification service which is proxied at the Gateway.
7. After order approvals from Andy, orders are sent to RC's Browserver. Having defined no intermediate Gateway, the success of the requests depend on its availability, so the server at TWP is set to only request its services on work hours.
8. At RC, Reno's browser displays the UI for the generic order service, with acceptance or rejection options.
9. Each notification is sent manually by Reno, executing the local notification service for each order.

Each notification message indicating the stage of the orders, is received by a notification service in each manager's Browserver and conveniently displayed as a means of tracking orders. After 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.

7. Conceptual Evaluation

The focus of this dissertation are the conceptual aspects of the Browserver over existing solutions. The evaluation compares the Browserver with widely used solutions, namely the usage of traditional W3C' Web Services applications and RESTful web services applications.

7.1 Methodology

To each conceptual aspect to evaluate, it is assigned a qualitative classification mark that indicates how well each solution suits it. Each classification only makes sense when compared with the same conceptual classification of the other solutions and not isolated from these.

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.

- null: none of the entities can start an interaction.
- low: both can start an interaction but none keeps context. Each interaction is completely independent.
- medium: only one of the entities can start an interaction, and only one keeps the interaction context.
- high: only one can start an interaction, both can keep interaction context.
- very high: both can start an interaction and maintain context.

(b) (In)Dependency. If the relationship is always dependent from a third party (external entity).

- null: the interaction is always dependent from a third party.
- low: an specific third party is needed to initiate and coordinate some of the interactions.
- medium: a specific third party is only needed to initiate the first interaction, then it is peer to peer.
- high: a generic third party is involved in part of the interactions.
- very high: no third party is needed.

(c) Presence. How the presence on the web of a person or organization (considering the persons that compose them) is managed.

- null: the presence is never guaranteed, even when the user is online.
- low: a user's (or organization) dedicated system is necessary to keep user's presence.
- medium: the presence is guaranteed by some application-specific remote provider.
- high: the presence is guaranteed with minor effort and without the need for a specific solution.
- very high: the user's presence is always transparently guaranteed.

(d) Privacy. How private remains the information shared. If the information is shared only between the participants of an interaction, the privacy is high, if otherwise any other entity has free access to it it is null.

- null: the information flows in the network and anyone can see it.
- low: the information is co-owned by a third party that intermediates the relationship, being the privacy conditioned by its policies.
- medium: intermediaries receive and forward messages and do not inspect their contents neither store information.
- high: security algorithms protect the information from any third party.
- very high: the information is guaranteed to be sent directly from one peer to the other. Security algorithms might also apply.

(e) Interaction patterns. If the interaction patterns enable to model any business process between any two entities it is very high, if it highly restricts the modelation, it is low.

- null: no interaction is possible.
- low: only synchronous interactions are possible.
- medium: only asynchronous unidirectional and bidirectional interactions are possible, provided that a communication channel between both can be created when a message needs to be sent.
- high: all synchronous interactions are possible when both peers are available and asynchronous interactions can occur even in the absence of one.
- very high: all synchronous and asynchronous interactions are possible even in the absence of the requested peer.

To be understood, some aspects depend on the context, which refers to the type of relationship that that two entities develop. 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. For this, consider scenario 1 described in Chapter 6.

Business to Business: Complex relationships between two organizations with common or distinct goals. Consider both scenarios described in Chapter 6.

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

7.2 Evaluation

Table 7.1 summarizes the comparative evaluation of solutions comprising W3C Web services, RESTful Web Services and Web Services developed with the Browserver as a platform.

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, being generic and universal is present in almost every user's computer, so RESTful solutions addressing the browser were considered.

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 being 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

Table 7.1: 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.

of existing solutions and lower the load on enterprise servers.

8. 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 as 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. In [Laclavik 08] a platform that uses existing email systems and sits either at the client or at the server side and is used to support business tasks by giving structure and improving the meaning of the information that is sent and received while managing its context with semantic annotations. Colayer [1] and Google Wave [2] contextualize all the communication and shared information into inter-related flows of information. The communication is always fully asynchronous and real time when participants are available. However, both of this platforms totally rely on dedicated and specific applicational servers that can have access to the information even if it is private and none of them empowers any entity with the capability of providing their own services (their capabilities are restricted by the services offered by the systems). Not only including the fully asynchronous communication model, the Browserver turns each user in a web service provider while shifting from client-server to a peer-to-peer paradigm. Moreover, the user himself gets to be seen and can be invoked a web service, therefore being able to interact with a business process in a complete service paradigm.

Smart Browser [Lin 10], intends to provide more processing power, enabling background processing that can change the way things are presented, extending the content that is presented on each retrieved Web page, but it does not give service provider capabilities to the user, who remains a mere consumer.

Opera Unite [3] couples a browser and a server, allowing the user to deliver content by request to other users. However, it does not follow a service paradigm and the communication is always proxied at Opera's servers. Moreover, if the user shuts down the application, all his resources 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 being possible to asynchronously request its services even if it is unavailable.

In [Matsubara 04] the author describes a method to get the user to be seen as a web services server to a business process engine. Although, the method results in sending a notification

[1] Colayer: http://www.colayer.com, last access on 2010-08-25
[2] Google Wave: http://wave.google.com, last access on 2010-08-25
[3] Opera Unite: http://unite.opera.com/, last access on 2010-08-25

(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, being the email the most used. The Browserver otherwise, enables direct requests 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.

Most of the existing P2P applications are centered at file sharing being BitTorrent the most used nowadays or for specific collaboration purposes, like MS SharePoint [4] with its shared workspaces also based on file sharing, consisting these on specific platforms for specific ends. 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.

The unification of human and electronic services is getting more attention nowadays, although there is no widely adopted solution for the problem. We argue that the solution lies on empowering the user with service provision capabilities in a complete service paradigm for modelling Web applications. Thhe authors in [Schall 08] and [Schall 09] seem to be committed to the problem with a Human Provided Services system, but lacking the universal platform that the Browserver aims to be, not giving providing capabilities to the user.

[Harrison 05] describes WSPeer as an interface to hosting and invoking web services, however it is not aimed user interaction but rather on creating a peer-to-peer platform for service deployment and discovery. [Mondejar 06] provides means to discover and deploy services in a structured P2P network. The services are replicated through several nodes enabling high availability, though the service location is not controllable by it's owner. The Browserver provides a 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 service is a decision of its owner and can be locally or remotely deployed.

[Song 07] and [Kassoff 03] 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.

[4] Microsoft SharePoint Workspace: http://office.microsoft.com/en-us/sharepoint-workspace-help/, last access on 2010-08-25

Most of current efforts to improve the Web are centered at the user experience as a consumer. The HTML5 draft [Hickson 10c] enables more interactive content, by extending the dynamic UI creation to meet the standards. Anyhow, many capabilities it will bring to the browsers can be done by the local server and, for greater user interaction, also Flash can be used, so the choice isnt limited.

Still in a draft state of an API [Hickson 10a] and a standard protocol [Hickson 10b], Web Sockets promise to enable seamless bi-directional communication and, consequently, much lower latency in connections between browser and server, even through intermediary proxies and firewalls (if encryption is used). The Browserver might eventually benefit from the use of such technology for communication, although the direction the technology is heading does not put the user in a provider position, as the services still remain at the servers.

9. Conclusions

This chapter summarizes the main conclusions of this dissertation and discusses the future work regarding the Browserver investigation.

9.1 Conclusions

This dissertation presented the Browserver as a human interface to web services and 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 step forward for the Internet of Services, and that the Browserver can be applied to business relationships in an inter-organizational environment.

As a result of this dissertation, providing services and direct interactions between users using automatically generated UIs for local and remotely located services is already possible, even though the implementation stage is at a prototype level. The architectural solution and implementation described in this document, represent descriptions of a Browserver that couples existing technologies for demonstration of the concept. A fully integrated light solution for the end user would consist of a new generation of browsers with an integrated web service server adapted to the Browserver needs.

The Browserver changes the way Web applications are designed. People, the leaves of the current Web, can be invoked as if they were Web Services and workflows can be implemented by knowing that each participant is able to perform a task and to provide a service that is directly requested (as in a real business process) and not relying on the user's willingness to follow an URL.

Using the Browserver, enterprises can integrate the electronic management of a supply chain with its partners even with little resources and small IT infrastructures (down to only one computer intermittently connected to the Internet). Through the Browserver and apart from its self provided business services, its user can be seen as a web service and generically invoked in a business process requiring is involvement. With two business scenarios in the background, it is shown how the Browserver can be used and conceptually evaluated it facing traditional solutions. Used either as a personal or as an enterprise application platform, the Browserver has to offer:

- 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.

9.2 Future Work

Although the main architectural solutions presented have already been implemented and tested, there is still a long way to go for a final Browserver. Not all the current technologies are well suited for the Browserver. While Web Services are still the most used standard technology to implement the service paradigm, their complexity and sluggish performance constitute an opportunity for alternatives that best suit performance and scalability, such as WOA and REST [Pautasso 08]. However, expressiveness is not the strongest point in REST. The convergence of the two approaches is now the focus of study and development [Schroth 07].

- Still as a work in progress, the Unified Services Model [Delgado 08a] and the Unified Services Implementation Language (SIL) represent a perfect fit for the Browserver. An architectural implementation with the SIL is must be made and compared with the one here presented.
- NAT constitutes an obstacle to P2P networks. Solutions for the problem exist, but were not object of study by this dissertation and must be addressed in the future.
- As high availability and performance is required for the Gateway, distributed solutions that take in account the necessary control over the location of each published document must be studied and tested.
- Service delegation is another concept that can play an important role in the continuous presence (or availability) of the services. Further investigation over this subject is required. Service delegation 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.
- Although the minimum set of services has already been defined, it is necessary to determine the full set of primitive services and operations for an operational basic Browserver in heterogeneous environments.
- The 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.

References

[B. Atkinson 02] S. Hada M. Hondo P. Hallam-Baker J. Klein B. LaMacchia P. Leach J. Manferdelli H. Maruyama A. Nadalin N. Nagaratnam H. Prafullchandra J. Shewchuk B. Atkinson G. Della-Libera & D. Simon. *Web services security (WS-Security)*. Rapport technique, April 2002. http://www-106.ibm.com/developerworks/library/ws-secure/, last access on 2010-10-14.

[Bellwood 04] Tom Bellwood, Steve Capell, Luc Clement, John Colgrave, Matthew J. Dovey, Daniel Feygin, Andrew Hately, Rob Kochman, Paul Macias, Mirek Novotny, Massimo Paolucci, Claus von Riegen, Tony Rogers, Katia Sycara, Pete Wenzel & Zhe Wu. *UDDI Version 3.0.2*, October 2004. www.oasis-open.org/committees/uddi-spec, last access on 2010-10-14..

[Berners-Lee 94a] T. Berners-Lee. *RFC 1738, Uniform Resource Locators (URL)*. Request For Comments (RFC), 1994. http://www.ietf.org/rfc/rfc1738.txt, last access on 2010-10-14.

[Berners-Lee 94b] Tim Berners-Lee, Robert Cailliau, Ari Luotonen, Henrik Frystyk Nielsen & Arthur Secret. *The World-Wide Web*. Commun. ACM, vol. 37, no. 8, pages 76–82, 1994.

[Berners-Lee 05] T. Berners-Lee, R. Fielding & 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.

[Christensen 01] Erik Christensen, Francisco Curbera, Greg Meredith & Sanjiva Weerawarana. *Web Service Definition Language (WSDL)*. Rapport technique, March 2001. http://www.w3.org/TR/wsdl, last access on 2010-10-14.

[Cox 02] Russ Cox, Athicha Muthitacharoen & Robert Morris. *Serving DNS Using a Peer-to-Peer Lookup Service*. In IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems, pages 155–165, London, UK, 2002. Springer-Verlag.

[Crane 08] Dave Crane & Phil McCarthy. Comet and reverse ajax: The next-generation ajax 2.0. Apress, Berkely, CA, USA, 2008.

[Delgado 08a] 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.

[Delgado 08b] J. Delgado. Unified services implementation language (sil). unpublished report, https://dspace.ist.utl.pt/bitstream/2295/581354/1/sil.pdf, last access on 2010-10-14., 2008.

[Du 06] Z. Du, J. Huai & 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.

[Erl 05] Thomas Erl. Service-oriented architecture: Concepts, technology, and design. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[Fielding 00] Roy T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[Flanagan 06] David Flanagan. Javascript: the definitive guide. 5 edition, 2006.

[Garrett 05] J. Garrett. *Ajax: A New Approach to Web Applications*, February 2005. http://www.adaptivepath.com/publications/essays/archives/000385.ph, last access on 2010-10-14.

[Hapner 02] Mark Hapner, Rich Burridge, Rahul Sharma, Joseph Fialli & Kate Stout. *ava Message Service (JMS) Specification v1.1*. Rapport technique, Sun Microsystems Inc, April 2002. http://java.sun.com/products/jms/docs.html, last access on 2010-10-14.

[Harrison 05] A. Harrison & 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.

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

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

[Hickson 10c] Ian Hickson & David Hyatt. *HTML 5: A Vocabulary and Associated APIs for HTML and XHTML*. W3C Working Draft, March 2010. http://www.w3.org/TR/html5/, last access on 2010-10-14.

[Hinchcliffe 08] Dion Hinchcliffe. *What Is WOA? It's The Future of Service-Oriented Architecture (SOA).*, 2008. http://hinchcliffe.org/archive/2008/02/27/16617.aspx, last access on 2010-10-14.

[Hoegg 06] Roman Hoegg, Robert Martignoni, Miriam Meckel & Katarina Stanoevska-Slabeva. *Overview of business models for Web 2.0 communities*. In Klaus Meibner & Martin Engelien, editeurs, Proc. Workshop Gemeinschaften in Neuen Medien (GeNeMe), pages 33–49, Dresden, 2006. TUDPress, Dresden.

[Iwasa 04] K. Iwasa. *Web Services Reliable Messaging TC WS-Reliability 1.1, OASIS*. Rapport technique, Fujitsu Limited, Novell, Inc., Oracle Corporation, Sun Microsystems, 2004. http://docs.oasis-open.org/wsrm/ws-reliability/v1.1, last access on 2010-10-14.

[Jacobs 99] Ian Jacobs, David Raggett & Arnaud Le Hors. *HTML 4.01 Specification*. W3C recommendation, W3C, December 1999. http://www.w3.org/TR/1999/REC-html401-19991224, last access on 2010-10-14.

[Kassoff 03] Michael Kassoff, Daishi Kato & Waqar Mohsin. *Creating GUIs for Web Services*. IEEE Internet Computing, vol. 7, pages 66–73, 2003.

[Laclavik 08] Michal Laclavik, Martin Seleng, Emil Gatial & Ladislav 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.

[Lantz 03] A. Lantz. *Does the use of e-mail change over time*. International Journal of Human-Computer Interaction, vol. 15, pages 419–431, June 2003.

[Lassila 07] Ora Lassila & James Hendler. *Embracing Web 3.0*. IEEE Internet Computing, vol. 11, no. 3, pages 90–93, 2007.

[Lin 10] Demiao Lin, Jianming Jin & Yuhong Xiong. *Smart Browser: A framework for bringing intelligence into the browser*. volume 7540, Tower A505 SP

Tower, Tsinghua Science Park, HaiDian District, Beijing, China, 100084, 20010.

[Marginian 09] D. Marginian & J. Walke. *Direct Web Remoting - Easy Ajax forJava*, 2009. http://directwebremoting.org, last access on 2010-10-14.

[Martin 04] David Martin, Mark Burstein, Erry Hobbs, Ora Lassila, Drew Mcdermott, Sheila Mcilraith, Srini Narayanan, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan & Katia Sycara. *OWL-S: Semantic Markup for Web Services*. Rapport technique, November 2004. http://www.w3.org/Submission/OWL-S/, last access on 2010-10-14.

[Matsubara 04] Daisuke Matsubara & Kazuho Miki. *Method and Apparatus for Peer-to-peer access*. Patent US 2004/0148434 A1, Jul 2004.

[McCabe 04] Francis McCabe, David Booth, Christopher Ferris, David Orchard, Mike Champion, Eric Newcomer & Hugo Haas. *Web Services Architecture*, February 2004. http://www.w3.org/TR/ws-arch/, last access on 2010-10-14.

[Mondejar 06] Ruben Mondejar, Pedro Garcia, Carles Pairot & Antonio 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.

[Nielsen 03] Henrik F. Nielsen, Noah Mendelsohn, Jean J. Moreau, Martin Gudgin & Marc 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.

[O'Reilly 07] Tim O'Reilly. *What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software*. MPRA Paper 4578, University Library of Munich, Germany, March 2007.

[Pappas 06] V. Pappas, D. Massey, A. Terzis & L. Zhang. *A Comparative Study of the DNS Design with DHT-Based Alternatives*. In In the Proceedings of the 25th IEEE Conference on Computer Communications (INFOCOM), 2006.

[Pautasso 08] Cesare Pautasso, Olaf Zimmermann & Frank Leymann. *Restful web services vs. "big"' web services: making the right architectural decision*. In WWW '08: Proceeding of the 17th international conference on World Wide Web, pages 805–814, New York, NY, USA, 2008. ACM.

[Ramasubramanian 04] Venugopalan Ramasubramanian & Emin Gün Sirer. *The design and implementation of a next generation name service for the internet*. In SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications, pages 331–342, New York, NY, USA, 2004. ACM.

[Rescorla 00] E. Rescorla. *RFC 2818: URN syntax*, May 2000. http://www.ietf.org/rfc/rfc2818.txt, last access on 2010-10-14.

[Resig 10] John Resig. *jQuery: The Write Less, Do More, JavaScript Library*, 2010. http://jquery.com, last access on 2010-10-14.

[Richardson 07] Leonard Richardson & Sam Ruby. Restful web services. O'Reilly Media, 2007.

[Rowstron 01] Antony Rowstron & Peter Druschel. *Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems*. Lecture Notes in Computer Science, vol. 2218, 2001.

[Schall 08] Daniel Schall, Hong-Linh Truong & Schahram Dustdar. *Unifying Human and Software Services in Web-Scale Collaborations*. IEEE Internet Computing, vol. 12, no. 3, pages 62–68, 2008.

[Schall 09] Daniel Schall, Christoph Dorn, Hong-Linh Truong & Schahram Dustdar. *On Supporting the Design of Human-Provided Services in SOA*. pages 91–102, 2009.

[Schroth 07] Christoph Schroth & Till Janner. *Web 2.0 and SOA: Converging Concepts Enabling the Internet of Services*. IT Professional, vol. 9, pages 36–41, 2007.

[Schumacher 07] Michael Schumacher, Tim van Pelt, Ion Constantinescu, Alexandre de Oliveira e Sousa & Boi 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.

[Song 07] Kisub Song & Kyong H. Lee. *An Automated Generation of XForms Interfaces for Web Services*. IEEE International Conference on Web Services, vol. 0, pages 856–863, 2007.

[Stoica 01] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek & H. Balakrishnan. *Chord: A scalable peer-to-peer lookup service for internet applications*. Proceedings of the 2001 SIGCOMM conference, vol. 31, no. 4, pages 149–160, 2001.

[van Kesteren 09] Anne van Kesteren. *XMLHttpRequest*, November 2009. http://www.w3.org/TR/2009/WD-XMLHttpRequest-20091119/, last access on 2010-10-14.

[Venkatapathy 02] C. Venkatapathy & 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.

[W3C 09] W3C. *HTTP - Hypertext Transfer Protocol.*, November 2009. http://www.w3.org/Protocols/, last access on 2010-10-14.

[Weerawarana 05] Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey & Donald 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.

[White 04] S.A. White. *Introduction to BPMN*. Rapport technique, 2004.

[Whittaker 96] Steve Whittaker & Candace 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.

Appendices

A User Interface XML Schema Definition

Listing 1 presents the XML Schema for the user interface definition set for the Browserver prototype implementation of this dissertation.

Listing 1: XML Schema for Defining User Interfaces

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >

  <xs:element name="interface" type="interfaceType"/>

  <xs:complexType name="interfaceType">
    <xs:group ref="iGroup"  minOccurs="0" maxOccurs="unbounded"/>
    <xs:attribute name="title" type="xs:string"/>
    <xs:attribute name="id" type="xs:string"/>
  </xs:complexType>

  <xs:group name="iGroup">
    <xs:sequence>
        <xs:choice>
          <xs:group ref="fGroup"/>
          <xs:element name="form" type="formType"/>
        </xs:choice>
    </xs:sequence>
  </xs:group>

  <xs:group name="fGroup">
    <xs:sequence>
        <xs:choice>
          <xs:element name="link" type="linkType"/>
          <xs:element name="action" type="actionType"/>
          <xs:element name="image" type="imageType"/>
          <xs:element name="p" type="paragraphType"/>
          <xs:element name="text" type="textType"/>
          <xs:element name="group" type="groupType"/>
          <xs:element name="table" type="tableType"/>
        </xs:choice>
    </xs:sequence>
  </xs:group>
```

```
  <xs:complexType name="tableType">
      <xs:sequence>
          <xs:element name="row" minOccurs="0" maxOccurs="unbounded">
              <xs:complexType>
                  <xs:choice>
                      <xs:element name="cell" type="cellType"
                                  minOccurs="1" maxOccurs="unbounded"/>
                      <xs:element name="hcell" type="cellType"
                                  minOccurs="0" maxOccurs="unbounded"/>
                  </xs:choice>
                  <xs:attribute name="type" type="bgType"/>
              </xs:complexType>
          </xs:element>
      </xs:sequence>
  </xs:complexType>
  <xs:simpleType name="bgType">
      <xs:restriction base="xs:string">
          <xs:enumeration value="default"/>
          <xs:enumeration value="separator"/>
          <xs:enumeration value="highlight"/>
      </xs:restriction>
  </xs:simpleType>

  <xs:element name="cellType">
      <xs:complexType>
          <xs:attribute name="colspan" type="xs:integer"/>
          <xs:attribute name="rowspan" type="xs:integer""/>
          <xs:attribute name="pos" type="posType""/>
      </xs:complexType>
  </xs:element>

<xs:complexType name="groupType">
  <xs:group ref="iGroup"  minOccurs="0" maxOccurs="unbounded"/>
</xs:complexType>

<xs:complexType name="actionType">
  <xs:sequence minOccurs="0" maxOccurs="1">
        <xs:element name="p" type="xs:string"/>
        <xs:element name="text" type="xs:string"/>
        <xs:element name="panel" type="panelType"/>
```

```
  </xs:sequence>
  <!—<xs:group ref="fGroup"  minOccurs="0" maxOccurs="unbounded"/>—>
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="type" type="xs:string"/>
  <xs:attribute name="confirm" type="xs:string"/>
</xs:complexType>
<xs:simpleType name="actionTypeType">
  <xs:restriction base="xs:string">
      <xs:enumeration value="box"/>
      <xs:enumeration value="button"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="panelType">
  <xs:sequence>
        <xs:element name="action" type="actionType"
                    minOccurs="1" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="title" type="xs:string"/>
</xs:complexType>

<xs:complexType name="formType">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:choice>
        <xs:element name="input" type="inputDataType"/>
        <xs:element name="select" type="selectType"/>
        <xs:group ref="fGroup"/>
      </xs:choice>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="buttonText" type="xs:string"/>
</xs:complexType>

<xs:complexType name="selectType">
  <xs:choice>
    <xs:element name="option" maxOccurs="unbounded">
      <xs:complexType>
          <xs:simpleContent>
              <xs:extension base="xs:string">
                  <xs:attribute name="value" type="xs:string" use="required"/>
```

```
                </xs:extension>
            </xs:simpleContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="knownProviders" maxOccurs="1">
        <xs:complexType>
            <xs:complexContent>
                <xs:restriction base="xs:string">
                    <xs:attribute name="service" type="xs:string" use="required"/>
                </xs:restriction >
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
  </xs:choice>
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="type" type="selectTypeType"/>
  <xs:attribute name="selected" type="xs:string"/>
</xs:complexType>

<xs:simpleType name="selectTypeType">
  <xs:restriction base="xs:string">
      <xs:enumeration value="dropdown"/>
      <xs:enumeration value="radio"/>
  </xs:restriction>
</xs:simpleType>

  <xs:element name="inputDataType">
      <xs:complexType>
          <xs:simpleContent>
              <xs:extension base="xs:string">
                  <xs:attribute name="name" type="xs:string" use="required"/>
                  <xs:attribute name="value" type="xs:string" use="required"/>
                  <xs:attribute name="type" type="inputDataTypeType"/>
                  <xs:attribute name="width" type="xs:positiveInteger"/>
                  <xs:attribute name="height" type="xs:positiveInteger" />
                  <xs:attribute name="disabled" type="xs:Boolean"/>
              </xs:extension>
          </xs:simpleContent>
      </xs:complexType>
  </xs:element>
```

```
<xs:simpleType name="inputDataTypeType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="text"/>
        <xs:enumeration value="textarea"/>
        <xs:enumeration value="file"/>
        <xs:enumeration value="date"/>
        <xs:enumeration value="radio"/>
    </xs:restriction>
</xs:simpleType>

<xs:complexType name="imageType">
    <xs:attribute name="name" type="xs:string" use="required"/>
</xs:complexType>

<xs:complexType name="paragraphType">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="text" type="textType"
                        minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:complexType>

<xs:element name="linkType">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:string">
                <xs:attribute name="url" type="xs:string" use="required"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>

<xs:element name="textType">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:string">
                <xs:attribute name="weight" type="textWeightType"/>
                <xs:attribute name="size" type="textSizeType"/>
```

```
                </xs:extension>
            </xs:simpleContent>
        </xs:complexType>
    </xs:element>
    <xs:simpleType name="textWeightType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="normal"/>
            <xs:enumeration value="bold"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="textSizeType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="small"/>
            <xs:enumeration value="normal"/>
            <xs:enumeration value="big"/>
        </xs:restriction>
    </xs:simpleType>

</xs:schema>
```