Design and implementation of a generic per-fee-link framework

Antonio Ruiz-Martínez\textsuperscript{1}, Óscar Cánovas\textsuperscript{2} and Antonio F. Gómez-Skarmeta\textsuperscript{1}
\textsuperscript{1}Department of Information and Communication Engineering
\textsuperscript{2}Department of Computer Engineering, Faculty of Computer Science
30100 Campus de Espinardo, Murcia, Spain
\{arm, ocanovas, skarmeta\}@um.es

Abstract

\textbf{Purpose} – This contribution presents a viable approach for designing and implementing a generic per-fee-link framework. This framework has been designed to be used with any payment protocol and has been tested with two existing ones.

\textbf{Design/methodology/approach} – The paper presents a per-fee-link framework based on several generic components. These components have been developed and tested in order to prove the viability of the proposed framework.

\textbf{Findings} – Our results show that is possible to establish a per-fee-link framework. Four core components are defined: first, the different modules needed for browsers and web servers, second, an Extended Payment Protocol (EPP), which negotiates the payment protocol to use and encapsulates its related messages, third, an API for e-wallets, which is independent of the payment protocol, to incorporate the protocols to use with EPP and finally, the definition of a per-fee-link that associates payment information to a link.

\textbf{Practical implications} – The framework presented shows a uniform way of using payment protocols that can increase the trust of end users. Furthermore, it has been developed and tested.

\textbf{Originality/value} – The contribution describes the components needed for supporting the framework. Its feasibility has been checked through an implementation and it facilitates the payment for content on the Web. Thus, content providers can obtain an alternative revenue source to advertisement or subscription. Furthermore, developers, vendors and customers can see that the incorporation of payment protocols to the system is facilitated. Finally, the users obtain a uniform way to make payments that increases the perception of trust.

\textbf{Keywords} – Electronic Payment System, Electronic payment protocol, Internet, per-fee-link, e-wallet, Extended Payment Protocol

\textbf{Paper Type} – Research paper.

1. Introduction

Electronic commerce (e-commerce) is a research area that is returning to show a prominent future (Lin, 2008). This is due to the fact that companies need to provide an online shopping or marketing presence for their customers. In e-commerce a key field is electronic payment systems. The use of electronic payments (e-payments) is especially attractive for end users to pay the links visited or accessed of a web page (Hezberg, 2003; O'Mahony et al., 2001), i.e., for purchasing newspaper articles, research papers, MP3 files, etc. As commented in (Song et al., 2002), users prefer the use of e-payment instead of subscription to access to this kind of electronic content. The W3C has named this model payment per-visited-link or per-fee-links (Chang and Dardailler, 1997). This model is also very interesting for content providers because it has been proposed as an alternative revenue source to both advertisement in the Web and subscriptions as mentioned in (Dai and Grund, 2007; Kytojoki and Karpjoki, 2001; Meng and Zhang, 2005; Rivest, 2004; Schmidt and Müller, 1999).

At the present time, there is no successful generic framework for using electronic payments following the per-fee-links model. Vendors need this framework because existing models do not offer customers a secure and trusted way to make payments on the Internet. As mentioned in (Gefen et al., 2003), the basis to establish trust in on-line payments are: a) a belief that the vendor has nothing to gain by cheating b) a belief that there are safety mechanisms built in into the web site c) a user-friendly interface, and d) ease of use. However, the current deployments offering payment solutions mostly use their own proprietary methods. This fact provokes distrust in customers because the method of payment for each vendor is different. Therefore, users are not sure whether the procedure is secure enough. Furthermore, it follows that payment implementations from different vendors do not interoperate and thus the introduction of e-payments in the Web has been
slow. In spite of the fact there are a lot of secure payment protocols, in the end, most payments are performed by sending a credit card number with a SSL/TLS connection.

A framework supporting the abovementioned per-fee-link model should include the following requirements. Firstly, it should manage the choice or negotiation of the payment method, the brand and the protocol to use. Secondly, the framework should define the principles for a transport mechanism able to encapsulate the different messages of the protocol chosen to make the payment and receive the content (a wide range of existing payment protocols should be supported). Furthermore, this transport mechanism should be able to deal with payments of low value or very low value (micropayments) efficiently. It should also support repeated/ongoing payments (the means to specify that a payment covers not only the link purchased but also some limited number of future purchases) as well as an efficient method of making successive payments. Finally, the incorporation of a new payment method should be both well-defined and easy for both customers and vendors, i.e., with an Application Programming Interface (API) for e-wallets to support the framework.

In response to the requirements of the per-fee-link model, this paper offers the following:

- A complete solution that covers all the elements needed for supporting the per-fee-link model in a generic way (Section 2). This solution is the result of the combination of several elements that we describe throughout this paper (Sections 3, 4, 5 and 6). Each one of them improves previous works in this field as discussed in Section 8.
- Framework components for browsers and servers (Section 3).
- Most importantly, we present a new payment-independent transport mechanism (protocol) in Section 4. This protocol is independent of the application protocol used. Therefore, we can use it to pay for HTML pages, FTP files, etc. Furthermore, it supports options for the negotiation of the payment. This protocol is specially designed to support micropayments, but it can be used with any payment protocol. It is session-oriented and therefore, different payments may be made in the same session. Moreover, unlike other previous proposals, we can support not only the per-fee-links model but also other different models such as pay-per-a-set-of-links, pay-per-time, etc.
- The design of a generic e-wallet based on a finite state machine (Section 5) that allows us to support different payment protocols. Furthermore, the e-wallet supports the easy integration of any payment protocol into the framework. Thus, we facilitate the use these payment protocols by developers, vendors and clients.
- The method to associate payment information to a link in a web page, i.e., the definition of a per-fee-link (Section 6).
- The design and the implementation (Section 7) of a system based on the components that we have just mentioned. This paper also shows the viability of the proposed framework (Section 7). It is also important to point out that the implementation does not require the re-implementation of browsers and web servers because it is based on extension mechanisms.

2. Per-fee-links framework

In this section we specify the components of our framework and the payment process that is followed to make a payment for a per-fee-link.

Two main actors participate in our framework: a client using his browser and the vendor’s web server. Additionally, a Payment Service Provider (PSP – an entity that accepts payments on behalf of the vendor) could participate depending on the payment protocol used to make the payment. In general, this entity is not required for micropayments (Day and Grund, 2007; Kytojoki and Karpijoki, 2001; Meng and Zhang, 2005; Rivest, 2004; Schmidt and Müller, 1999).

The payment process is depicted in Figure 1. Let us suppose a user wants some content such as a paper (in HTML, PDF, …), an MP3, an image,… After surfing on the Net, he discovers a vendor that provides that content. Then, he enters/clicks the URL in the web browser (step 1). Next, the browser sends a request to the vendor’s web server (step 2) to obtain it (step 3). This page is shown in the browser (step 4). This page could contain one or more per-fee-links that offer the content requested, indicating that the content requires a payment. If the client is willing to pay for the content, he clicks on the per-fee-link (step 5). Next, the browser and server establish a session to negotiate the payment options and make the payment. Thus, they negotiate the payment protocols, brands and PSPs (from now on, we simply name a combination of these elements as the payment option) to use (step 6). The server, with the payment options, indicates the prices of the content using each payment option. It is important to point out that prices may vary depending on the payment option. The reason is that the cost of protocols is not always the same; it varies depending on the computational costs associated with the messages, the banks’ commissions, etc.

Once the payment options have been negotiated, the browser requests authorisation from the user to initiate the payment process (step 7). This step is optional depending on the user’s e-wallet configuration. For example, in
micropayments the user can decide to make an authorisation either for each five dollars/euros, or for each time he sends 5 microcoins,… If the user agrees to the payment, the payment protocol is executed (step 8). Depending on the payment protocol chosen, participation of a PSP could be required in the payment protocol (step 9). In general, it is the vendor who is connected to the PSP. However, there are payment protocols where it is the user who sends the payment to the PSP. When the payment is made, the server sends the URL that actually allows the browser (step 10) access to the content. The browser uses this new URL to access the content that has been paid for (step 11). Thus, the server provides the content (step 12). Finally, the content is shown to the user (step 13). Depending on the content, a different application could be launched (MP3 player, PDF reader, …).

Figure 1. Payment process.

When a user makes a payment to a vendor, this payment could provide access only to one item of content (e.g. one research paper), or it could give the right to access to several items (e.g. several research papers), depending on the business model. If the payment gives access to several items of content, in the following per-fee-links requests, the web server provides the URL to the content without requiring a payment. That is because we maintain a session between the browser and the server. Furthermore, the server can also use this session to request a payment at any subsequent moment. Thus, we can support other models such as payment per-volume-of-data, or payment for a time period. How we can support these models is explained in Section 4.2.

Four components are needed to support this per-fee-link framework.

1. It is necessary to define the different components that are added to both the client and vendor architecture.

2. As an important part of these components, a session-oriented protocol is required that allows the browser and the server to establish the payment options to use. This new protocol, which is named Extended Payment Protocol (EPP), also encapsulates the different messages of the payment protocol chosen. Finally, once the payment is made, we receive the access to the Web content. The EPP messages are exchanged over an SSL/TLS channel to guarantee the confidentiality and integrity of the payment information exchanged. This protocol is detailed in Section 4 and it could be used to obtain access, based on payment, to any resource independent of the application/transfer protocol (HTTP, FTP) used to recover it.

3. We need to define the API that any e-wallet should satisfy in order to work with EPP. This API is independent of the payment protocol used to facilitate the development of new payment protocol components that can be incorporated in a seamless way to this framework. This e-wallet has to manage three kinds of functions: session functions (open, close,…), query functions (payment options supported,…) and payment functions. More details about the different e-wallet functions are mentioned in Section 5.
4. We need to differentiate between free-access-links and per-fee-links in a web page. To indicate that a link is a per-fee-link, and therefore EPP is used, we propose, for each known protocol, the addition of the letter “P” before its name in the field protocol of a URL: PFTP, PHTTP,… respectively for FTP, HTTP. For example, we use PHTTP as the protocol name to indicate that the URL points a resource that requires a payment (sent using EPP protocol) and that is accessed by means of HTTP. Additionally, with the per-fee-link we could include the payment options supported as well as the different prices in the different currencies. If this information is not included with the link, it is exchanged using EPP. We include this information in HTML-associating links with XML or RDF statements in the HEAD section (see Section 6).

![Diagram of Browser and Web Server Architecture]

3. Browser and server architecture

In this section we describe the different components to be incorporated into a browser and a web server to support the proposed framework.

The framework extends the browser’s architecture with several components (see Figure 2a) which are described next:

- **One or more e-wallets.** A wallet is a module that encapsulates the behaviour of a particular payment protocol. The specification of these wallets is provided in Section 5.
- **A configuration e-wallet (config e-wallet) module** that allows the addition and removal of the different payment protocols that can be used to make the payment with EPP.
- **A preference’s module** to specify user preferences, such as the preferred payment options, the maximum and minimum amounts that the client is willing to pay with each payment mechanism, etc.
- **The EPP module** that implements the EPP protocol (EPP module) described in Section 4. This module depends on the Configuration e-wallet and preferences modules to choose the protocols that are used in the payment. Once the protocol has been chosen, the EPP module uses the e-wallet that implements it.
- **One or more protocol handlers.** We include a protocol handler for each application protocol based on payment. The handler invokes the EPP module to perform the payment, e.g., the PHTTP module shown in the Figure 2a is a protocol handler.

On the server’s side, the architecture is depicted in the Figure 2b. There are several modules for the usual protocols supported by a web server such as HTTP, SSL/TLS. There are several modules to support different programming languages for the Web such as PHP and Perl modules. Furthermore, to support our framework we have included the EPP module which implements the EPP protocol. This concept of the EPP module is equivalent to the EPP protocol handler mentioned above. The reason we have decided to change the name is to maintain the terminology that is commonly used to name the different components of a web server. Finally, the last component is a set of server e-wallet modules. The concept of the server e-wallet is equivalent to the e-wallet in the browser architecture. The main difference between an e-wallet and a server e-wallet is that the e-wallet could have a graphical interface to interact with the user. However, the server e-wallet does not require any interaction and all the configuration information is established in configuration files.

4. Extended Payment Protocol
We provide a protocol to exchange information about the different payment options as well as the exchange of the messages from the payment protocol chosen in the negotiation. This protocol is named Extended Payment Protocol (EPP) and is session-oriented. Next, we describe the different phases that take place in EPP and the messages exchanged. After that, we explain the different payment models that are supported by EPP. Finally, we discuss some security issues.

4.1. Phases

EPP is divided into a set of logical steps. Firstly, there is an Initialisation phase where the parties establish a new session. Basically, the browser requests access to the per-fee-link and informs the server of the payment options he supports. The server answers with the subset of protocols that he supports and the amounts to pay. Before the establishment of the session, the browser checks that the server accessed is a secure payment server. For this purpose, the browser uses the payment certificate module that is installed on it. If the server certificate is not among the different secure payment server certificates or not in the chain of certificate authorities that issue secure payment server certificates, the session is closed.

Next, the Payment phase starts: the browser and the server exchange messages of the payment protocol chosen by the client from vendor’s response. Here, EPP is only responsible for encapsulating the messages generated by the e-wallet of that payment protocol.

Thus, the session is established and the browser obtains the information requested when the client clicked on the per-fee-link. This phase is named Application data. Depending on the model (see Section 4.2), after the client gets the information, he could visit more per-fee-links without making a payment (pay-per-time, pay-per-volume, etc) or he could be requested to make additional payments by returning to the Payment phase. Finally, after accessing the content, the client can finish the session (End of session phase).

Each phase is composed of, at least, one message. But in some of them, several messages could be sent between the browser and the server. In general, each exchange between the server and the browser is based on the pattern of the request-response, which is ideal for making micropayments. Furthermore, all the specific messages of the payment protocol chosen are encapsulated using an EPP message that can contain any of the different payment protocol messages. This message is conveyed over an SSL/TLS channel.

4.1.1. EPP message

An EPP message is composed of the following fields:

EPP message = Code, SessionID, Identifier, Data

- The Code field identifies the type of EPP message. There are four possible values: Request, Response, Success and Failure. The Request and Response values indicate whether the message contains an EPP request or response respectively. Success value indicates that the session was established successfully. An unsuccessful completion of the session is signified with the Failure value.
- The SessionID is a session identifier established between the browser and the server. The Identifier field aids in matching a response with a request in a session.
- Finally, the Data field contains the specific requests and responses used in the different protocol phases (see Figure 3).

Next, we detail the different phases we have described previously. In each phase, we depict the content of requests and responses in the Data field. The full message is no longer described because the fields Code, Session and Identifier are common to all messages.

4.1.2. Initialisation

The browser initiates the protocol using an EPP request message called Init (see Figure 3). Its Data field contains:

Data(Init) = Version, SupportedPaymentInfo, URL
where the Version field indicates the version of EPP that is being used. The SupportedPaymentInfo field specifies the payment options supported by the browser. Finally, the URL field contains the per-fee-link that the client wants to access to.

As a response, the server sends an EPP response message called InitResponse whose Data field contains:

\[
\text{Data(InitResponse)} = \text{Version, [PaymentInfo]}
\]

where the Version field was already mentioned. Note that we use [Data] to indicate that this piece of Data is optional. Thus, the PaymentInfo optional field contains the subset of payment options contained in the SupportedPaymentInfo that the server supports. For each payment option, the server specifies the price in different currencies.

4.1.3. Payment

In this phase, the browser and server exchange the messages of the payment protocol that the client has chosen for the session. This protocol is chosen from the protocols proposed in the initialisation phase. Depending on the payment protocol, its first message is sent by either the browser or the server. We have defined some EPP messages to encapsulate in the payment messages: WaitForMessage, PaymentProtocolMessage and PaymentRequest. These messages can be exchanged in several ways (see Figure 3). The number of payment messages to exchange is not limited in the EPP protocol. This limit is only imposed by the payment protocol.

The WaitForMessage indicates to the recipient that the sender is waiting for a payment message from the recipient. For example, if we suppose that we have a payment protocol that is initiated by the server, the browser sends this message to the server to request the first message of that payment protocol. This EPP message contains:
Data(WaitForMessage) = timeout, [typeOfMessage]

where the timeout field indicates the time a party will wait for the answer. Optionally, in the field typeOfMessage, the party can send which message, from the payment protocol, is waiting for.

The message named PaymentProtocolMessage encapsulates the messages of the payment protocol chosen. This message contains:

Data(PaymentProtocolMessage) = Protocol, Message

where the Message field is a payment message from the payment protocol chosen. The Protocol field has the information about which payment protocol the message comes from.

When the payment is made, the session is established and the server sends an EPP message with the Success type. In this message, the server provides a URL which the client uses to access to the content requested.

4.1.4. Application data

The client accesses the content using the application protocol established in the URL obtained (HTTP, FTP ...). Once the client has accessed the content, the session could finish, or depending on the model, the client could be allowed to access other per-fee-links. When the session is established, the EPP used to request the access to the new content is named RequestNewContent. Its Data field contains:

Data(RequestNewContent) = [SupportedPaymentInfo], URL

where URL field indicates the per-fee-link the user wants to access. If the payment made in the previous phase gives rights to access to more than one link (models: pay-per-a-set-of-links, pay-per-time,...), the server sends a Success response that contains the URL of the resource to access, as mentioned above.

On the contrary, if the previous payment only gives access to only one link, the server sends a PaymentRequest message. It indicates that the server requires a new payment to access to the new content requested. Its Data field is:

Data(PaymentRequest) = PaymentInfo

where PaymentInfo was previously explained (see InitResponse message).

After this message, both client and server return to the PaymentPhase to exchange the messages of the payment protocol chosen and to get the access to the content.

4.1.5. End of session

Once the client has accessed the content, he can close the session. For this purpose, we define an EPP message named CloseSession. Thus, both browser and web server can release the resources associated to the session.

4.2. Payment models

As mentioned throughout the paper, our framework supports different business models. The model, and its associated information, can be specified both in the payment information associated to the link (see Section 6) and in the EPP negotiation phase (in the PaymentInfo field). Several business models could be specified: pay per each link, pay per a set of links, pay per time, pay per the amount of data, etc.

In general, vendors require a payment for each link. Therefore, the user initiates the session, makes the payment and, finally he obtains the content. After that, if the client requests a new link in the same session, he is requested to make a new payment (as depicted in Figure 3, second payment phase). However, the process is faster because the initialisation was already made. Otherwise, a new session is initiated.

However, both client and vendor may agree on the payment for a set of links. In this model, the client makes an initial payment that allows him to access a set of links (the number of links that the user can access is specified in the PaymentInformation field or with the link). The payment is made when the user accesses the first element of the set of per-fee-links. Thus, there is an initialisation phase, then a payment phase and after that an Application data phase. For the
following per-fee-links, there is no need for the payment phase and the user is immediately provided with the new link in the Application data phase. The control of the per-fee-links visited are carried out by the server.

Finally, the pay-per-time or pay-per-data models are similar to the last model mentioned. In this case, in the payment information is specified the range of time/data the user is required to pay for. Once the payment is made, during that period of time or if the amount of data is not exceeded, a new payment is not required. Both the control of the data and time is carried out by the server.

4.3. Security considerations

From the security point of view, it is important to point out that the security of the payment phase depends on the payment protocol chosen since our proposal only includes the messages of the protocol chosen.

We have defined the exchange of EPP messages on top of SSL/TLS for the purpose of preventing some third party from seeing or changing the payment information exchanged between the browser and the server. For example, a third party could be interested in modifying the information about payment protocols so that both server and client choose a weak payment protocol that can be attacked. With SSL/TLS, we guarantee confidentiality and authentication of the server so that the client is sure he is paying the desired server.

Furthermore, for the payment servers that are trusted as well as the Certification Authority (CA) that are considered trusted, we follow a similar approach to the one followed by the web browsers so that a client can specify in his web browser the payment server in which he trusts. Thus, the web browser can automatically reject making a payment to a payment server that is not trusted or with a bad reputation or to those servers in which the client had problems previously.

5. Generic e-wallet

Apart from the EPP protocol, in order to be able to incorporate a payment protocol in both the web browser and the web server at any time without making modifications in them, we need to define a generic e-wallet or an e-wallet with a generic interface that can be used to support any existing or future payment protocols, as mentioned in Section 3. Thus, we have a common interface for all the payment software developers. This common interface facilitates the extensibility of the system as well as the trust of the users in the framework. This is due to the fact that, as mentioned in (Gefen, 2003), the method used to incorporate a new payment into both the web browser or the web server would be the same. This model is being used successfully in web browsers to incorporate other components like the support of cryptography by means of hardware devices, such as smart cards.

Our goal is to model the different payment components in a similar way. For this purpose we have reviewed the main payment protocols appearing in the literature (Millicent, SET, Paycash, etc). The description of the main payment protocols can be found in (O’Mahony et al., 2001). Next, we have analysed the different proposals of APIs that have appeared to date, like in SEMPER or in IOTP. These APIs, from our point of view, are not generic enough, and they do not cover some aspects like the management of sessions that reduces the information to exchange and makes the payment faster.

Conceptually, in any payment protocol there is an exchange of messages between two or more parties. In general, the participating entities are the client and the vendor, but other entities could participate such as a PSP, a bank or a broker. Thus, the design of the e-wallet is made taking into account that this is the main exchange. When other entities participate in the payment, the e-wallet is responsible for including the messages from other parties.

We can think about the e-wallet as a black box that receives messages. As a response to these messages, the e-wallet generates one or several messages. Thus, in a specific moment, when the e-wallet receives or answers a message, the following action to carry out can be modelled with four states: create a payment message as an answer, wait for a new message, wait for an event and the finalisation of the protocol in a suitable way or with an error in the execution of the protocol. Therefore, the main functions that any e-wallet should satisfy are: initialisation, receive a message, send a message, payment error and end of payment. As a result, any of these functions returns the following state for the e-wallet as well as the function to be invoked. Each state and function is represented in Figure 4. In this Figure, each state represents a function and each transition represents the result of invoking that function. This group of functions has been named payment functions. However, an e-wallet also needs other kinds of functions, such as functions related to the management of the session as well as some functions to query the payment options supported, the session, etc.

The session management functions are used to establish a session with the specific e-wallet chosen to make the payment so that we can make one or more payments by using the payment options supported by the e-wallet. For this purpose, we have defined an init session function that allows the initialisation of the e-wallet. When the session is initiated,
an identifier associated to the session is obtained. This identifier is used to distinguish between the different sessions that the browser can have, in parallel, with the e-wallet. The identifier is also used to invoke the rest of the functions that are part of the API. Another function related to a particular session is the initialisation of a particular payment mechanism (payment instance init function).

We have also defined several functions to query information about the different parameters related to the payment options supported by the e-wallet. We have defined a function to obtain the different brands supported, another function to obtain the payment protocols supported, another to query the different currencies supported by the different entities and protocols as well as another function to learn the current balance that the user has available to make payments.

![Finite state machine of the payment process.](image)

**Figure 4.** Finite state machine of the payment process.

### 6. Per-fee-link

In this section, we describe how to define per-fee-links in an HTML page. When the link is clicked, the payment process is launched. This payment process is based on the EPP protocol to convey the messages of the chosen payment protocol. Thus, we can support the per-fee-link model. The functionality of this protocol is incorporated into the web browser by a new plug-in or protocol handler which is launched when the user clicks on a URL that references this new protocol. The different payment modules (e-wallets) that a client can support are registered in the EPP module (see Section 3). In order to be able to incorporate any payment module, the e-wallets are implemented according to a generic API that we proposed in the previous section.

In HTML we propose to reference to this protocol by means of the URLs. This protocol is referenced by adding a “p” to each known protocol we are interested in to support payments. For example, to reference a web page that is accessed by means of payment, we reference it using the chain phttp://resource (it executes EPP to make a payment for a resource that is recovered using HTTP). Each new protocol is associated with a protocol handler in the web browser. This protocol handler is responsible for executing the EPP protocol, making the payment using the suitable e-wallet. After the payment is made, it passes the URL to the web browser in order to access the content. Optionally, in a URL we could include information about payment protocols and brands. If this information is not associated with the URL, the client cannot know the protocol supported to make the payment until the initialisation phase of EPP is executed. In the web page, this payment information is included in the HEAD section of HTML using XML or RDF statements. The URL references to this information by means of an id attribute in the A tag. The process followed to define a link based on payment is the following:

1. We include a link in the web page referencing a link that is based on payment. For this purpose we use the A tag. Then, in the href attribute of A tag we include the URL of the content that is based on payment. e.g. `<A href="phttp://resource">`.
2. We identify unequivocally the URL by means of the id attribute in the A tag.
3. Then, in the HEAD section of the HTML page, we include payment information referencing this link by means of the id attribute. This information is expressed by means of XML or RDF. The XML or RDF statements indicate the payment protocols supported, the brands supported and the PSPs supported as well as the price for each combination of these elements.
Next we show an example of a per-fee-link for this paper (Figure 5). Basically, this link requests a payment for a PDF file that represents an EPP paper. The seller of this paper is Research Papers and this link costs 15 Euros by using the SPEED protocol. The payment will be made to a PSP named Fast PSP.

When a per-fee-link is clicked, the protocol handler is launched and the EPP protocol is executed. The EPP module uses the e-wallets to make the payment. Once, the payment protocol is made, the EPP module obtains a URL that is shown by the browser. More details about an implementation can be found in the following section.

The definition of the different elements needed to express payment information was made by means of XML. We created an XML Schema that provides elements to describe payment brands, payment protocols, model to follow, prices, etc. This schema also defines the different elements that are part of the EPP protocol. From the schema, we have generated the source code needed to process these elements as we explain in the following section. Additionally, in order to improve the description of the payment information, we can annotate these elements with semantic information (Kopecký et al., 2007). The incorporation of semantic information facilitates the discovery of information based on intelligent agents or automatic processing.

Figure 5. Example of a per-fee-link.
7. Implementation and evaluation

In this section we discuss some issues about the feasibility of the implementation of the framework proposed. Our aim is to evaluate whether our proposal can be used with different payment protocols by means of the different components we have presented in the framework. It is expected that the time execution of our framework is higher than using the protocols directly. We only want to measure the increase and to verify if it is possible to assume this increase in order to have a general framework that facilitates payments on the Web to the different parties.

It is important to point out that our proposal does not require the re-implementation of all the browsers in order to be supported. On the contrary, our proposal can be included in the most well-known browsers and web servers because they offer some generic extensibility mechanisms to support new protocols. This increases acceptability.

The support of a new protocol in Firefox is included by defining what is known as a protocol handler extension, whilst in Internet Explorer the support of a new protocol is included by means of what is known as asynchronous pluggable protocols. On the other hand, from the point-of-view of web servers, one of the most well-known web servers, the Apache Web server, can be extended by what is known as modules. Indeed, the support of SSL in this web server is implemented as a module. In a similar way, our protocol would be implemented as a connection-level filter module.

Additionally, in order to prove the viability of our proposal, we have implemented and tested EPP. For this purpose, we also made use of the implementation of two payment protocols: Millicent (Glassman et al., 1995) and SPEED (Ruiz et al., 2001). Millicent is a micropayment protocol that is suitable for paying for content of very low value such as one cent or even less. SPEED is a stronger protocol that guarantees non-repudiation. The delivery of the product is ciphered and was designed to pay with smart cards.

Both implementations were developed in C++ under a Linux operating system. The messages of both protocols are expressed in ASN.1 because, before the success of XML, all protocols were defined by means of ASN.1. We used the OpenSSL library for ASN.1-related functions and as the cryptographic library.

For EPP we have also created a C++ library. From this library, we have created a client module. We have decided to support the server as a daemon tool instead of developing a server module in order to simplify the process of development. The EPP messages are expressed in XML.

We have performed the different tests on a laptop with the following configuration: Intel Core 2, 1.83Ghz, 2 GB of RAM and Fedora Core 6 as the operating system. Next, we are going to comment the different tests. Each test was repeated 100 times and the results presented in the next figures are based on mean times.

In Figure 6a, we compare the execution time of the Millicent payment protocol with the execution time of encapsulating the messages of Millicent in the EPP payment phase. We are only comparing this phase in order to estimate the overload introduced by an ASN.1 implementation of EPP in the payment. We do not compare the execution time with the whole execution because Millicent does not support negotiation of payment options or product delivery. In this situation the overhead introduced by EPP is only 5%.

In Figure 6b, we compare the difference between the implementation of the EPP Payment phase both in ASN.1 and in XML. As we can see, the ASN.1 implementation is 152 times better than the XML implementation. However, it represents an increase of only 40000 microseconds. Therefore the delay is acceptable from a user’s point-of-view.
From this analysis we can derive some conclusions. First, the development made in ASN.1 is far more efficient than the one based on XML. There are several reasons already mentioned in (Mundy and Chadwick, 2003): the ASN.1 element data size can be three times shorter than the XML one; in transmission time, the time deficit of XML can rise to 13.5 times for an element and the time of encoding/decoding data can reach up to ten times. Then, if we take into account that each EPP message is composed of several elements, we can understand this important difference in performance. Another reason for this considerable difference is that the information exchanged by the Millicent protocol is only a few bytes. However, when the size of the messages of the payment protocol is greater, this difference is reduced as we will see below.

The payment process is analysed next. In Figure 7a, we depict the comparison between the time of executing SPEED and the time of executing the initialisation and payment phases of EPP with SPEED. Then, in Figure 7b, we show the time of executing EPP with SPEED, indicating the time spent in each phase. We have not shown the end session phase because it is insignificant. It is also important to mention that with SPEED, the Application data phase is not necessary because the product is delivered with the payment. In Figure 7a, we can appreciate that the overhead of EPP with SPEED is maintained as long as the size of the product is not increased considerably (for the first product, EPP with SPEED is only 0.25 times slower than SPEED. However, for the last product, EPP with SPEED is ten times slower than SPEED). The constant increase of the EPP time in Graph 6a is due to the fact that any payment message has to be encoded in Base64 to be inserted in XML and in one of these messages, the product is sent ciphered. Thus, the bigger the product the higher the overhead. This fact can produce scalability problems when there are many concurrent requests. In spite of these data, enterprises are using XML and web services intensively (Erl, 2004; Jiang and Wliley, 2005), at different levels, for different type of services and purposes as mentioned in (Papazoglou and Georgopoulos, 2003) because “they provide a distributed computing infrastructure for both intra- and cross-enterprise application integration and collaboration”. In order to mitigate scalability problems, some solutions have been proposed such as the use of replication techniques, load-balancing servers and cluster-solutions. Thus, we might achieve a major availability and a better performance as proposed in (Birman, 2004; Mundy and Chadwick, 2003). Other additional options that try to cope with performance are web services based on ASN.1 (Fast Web Services) (Sandoz et al., 2003), compressed XML (Tian et al., 2003) or XML parsers based on streaming (Zhang and Engelen, 2006). In spite of these solutions, if the performance were critical to interoperability, then we could use the ASN.1 implementation that only supposes a 5% increase in the performance, as outlined above (see also Figure 6a).
From these results we can derive several conclusions. First, the EPP *Initialisation* and *End of session* phases introduce an almost constant overhead in microseconds because the information exchanged, i.e., the payment protocols negotiated, is the same for all the products. This overhead is lower as the time of the *Payment* phase increases, as we can see in the last case of Figure 7b. We cannot avoid this overhead because we need to establish the payment protocol to be used. Second, the performance of the XML implementation of EPP is stable as long as the size of the product does not increase considerably. We have mentioned the problems that could appear in the implementation of XML compared to ASN.1 and several solutions to cope with them. Finally, we can assert that it is possible to develop this framework to support the use of different payment protocols to pay for content on the Web. There were no problems in developing the EPP protocol as well as the two payment protocols as defined in the framework. As expected, the execution-time is higher than when using the different protocols directly. However, with the different tests we have performed, it is shown that the different parties could accept the overhead introduced.

8. Related research work

In this section we comment on some previous interesting work related to payment frameworks and frameworks for payment per-visited-links. To the best of our knowledge, there is no similar proposal to ours that covers all the aspects mentioned previously. Nevertheless, the following proposals cover some of the elements introduced in our framework.

JEPI (Chung and Dardaillet, 1997) proposes a payment-independent framework. They were the first to address the idea of a generic protocol. The main problem with this proposal is that the payment-options negotiation requires that the user surfs along several web pages to exchange that information. Furthermore, the use of payment protocols is limited only those that exchange only two messages (e-coins, microcoins,…).

SEMPER (Lacoste et. al, 2000) defines an API to develop e-wallets in a generic way. We have started from their ideas to develop our e-wallet. We have also taken into account that their proposal neither defines the messages and the information to be exchanged nor does it cover either different PSPs or session management.

W3C micropayment per-fee-links (Michel, 1999) is centred on defining the data required to insert the per-fee-links in HTML. The ideas of this proposal were useful to define our per-fee-link. In fact, we have followed their model. However, this proposal does not define how to convey the data and it supports neither multiple currencies nor different versions of a payment protocol. Finally, it does not handle payment methods that require the exchange of several messages to make a payment. Therefore, we have extended their results to take these issues into account.

IOTP (Burdett, 2000) offers an interoperable framework for Internet Commerce. It defined a generic payment protocol similar to that presented here. They also defined the information to negotiate the payment protocols, the information to include about payment protocols, etc. In our proposal, we have complemented and improved some of their elements to use them both in the EPP protocol as in the per-fee-link. However, IOTP does not support repeated/ongoing payments (Eastlake 3rd, 2002) and the API is too complex because is not only intended to make payments but also to carry out other aspects such as refunds, etc. Finally, it does not define how to include the payment information in the web page. These issues are covered in our proposal.

Finally, Meng’s proposal (Meng and Zhang, 2005) is based on a server e-wallet to provide multiple payment systems. It is an interesting idea that your e-wallet, supporting multiple payment systems, is in a server that can be accessed to make payments. However, a third party is involved in all transactions. This is not useful for micropayments, because of computational cost. Another aspect that this proposal does not cover is the per-fee-links approach.

9. Conclusions

We have presented a per-fee-link framework that is independent of the payment protocol but it is specially designed to support micropayments since these are the ideal kind of payment for low-value content. Our proposal is based on a session-oriented protocol that allows us to develop several business models: per-fee-links, pay-per-volume, etc. Furthermore, this protocol could be used to pay for any content that can be referenced by means of a URL. This protocol is supported by means of a protocol handler in the browser. This handler uses a different e-wallet for each payment protocol. However, the interface for each e-wallet is the same for any payment protocol and it is based on the abstraction of each protocol as a finite state machine. We have also mentioned how this protocol could be referenced in HTML as well as the functions that an e-wallet should provide. Therefore, we provide a comprehensive solution to answer to the needs of the per-fee-link model.

Furthermore, our proposal does not require the re-implementation of all the browsers and web servers. On the contrary, it can be included in the most well-known browsers and web servers. Moreover, we have described the details of an
implementation we have developed to prove the system proposed. Finally, this implementation has allowed us to test the system with two payment protocols. The results show that it is feasible to develop this framework and that the overhead introduced could be acceptable for the different parties.

Acknowledgements

This work has been partially funded by "Programa de Ayuda a los Grupos de Excelencia de la Fundación Séneca 04552/GERM/06".

References


