Payment Support in Ubiquitous Computing Environments

P. Boddupalli[1, 2], F. Al-Bin-Ali[1, 2], N. Davies[1, 2], A. Friday[2], O. Storz[2] and M. Wu[2]

[1]*Department of Computer Science*
University of Arizona
Arizona, United States

[2]*Computing Department*
Lancaster University
Lancaster, England

{bprasad, albinali, nigel}@cs.arizona.edu, {adrian, oliver, maomao}@comp.lancs.ac.uk

Abstract

Despite ten years of extensive research, Weiser's vision of ubiquitous computing is far from a widespread reality. While research into enabling technologies has progressed significantly, there has been a lack of research into the economic and commercial aspects of ubiquitous computing. In this paper, we investigate techniques that will enable investors in ubiquitous computing infrastructure and services to recoup their investment through charging for service use. In particular, we identify the key requirements for a ubiquitous computing payment system, discuss existing payment systems and present the Payment Session Protocol (PSP) that is designed to support payment-enabled ubiquitous computing environments.

1. Introduction.

Since the publication of Mark Weiser's influential paper on ubiquitous computing in 1991 [1], researchers have been working towards the realization of his vision of environments saturated with computing and communication capabilities. To date, the emphasis has been on fundamental research into enabling technologies such as networking, data management, security and user interfaces. As a result, prototype ubiquitous computing environments are being created within the laboratory [2][3][4]. However, the widespread deployment of ubiquitous systems is still far from reality and significant research challenges in the field of deployable ubiquitous computing systems remain [5].

One important element that will be critical in the widespread deployment of ubiquitous computing systems is a sound economic model that will encourage individuals and companies to invest in ubiquitous computing technology. The deployment of ubiquitous applications and environments is likely to involve huge investments and hence, an important precondition for such investment is that a suitable return can be obtained. This is in contrast to the 'Internet', which in its infancy was chiefly supported by government organizations. Moreover, while services such as mobile communications are typically controlled by a single operator, it is our belief that ubiquitous computing environments will thrive due to the input of many companies and individuals.

Drawing on the experiences of the Internet, we can identify a number of potential approaches to generating returns on investment in ubiquitous computing infrastructure and service provision: cross-subsidization, advertisement revenues, public investment and revenue collection from the end-user.

Cross-Subsidization:

In cross-subsidization models, the cost of service provision is covered by revenue that is generated through the use of other charged services. For example, consider a ubiquitous computing environment in a shopping mall that lets shoppers send email and use large displays. In the cross-subsidization model, the cost of providing these services could be borne by the outlets in the mall, which in turn pass the cost on to shoppers in the form of higher in-store prices. While examples of cross-subsidization can be found in many areas of commerce, we believe that this scheme is unlikely to provide a comprehensive solution for ubiquitous computing for a number of reasons. Firstly, infrastructure in ubiquitous computing environments is likely to be owned by more than one individual or organization: in the shopping mall, different companies may provide the displays, email server, and communication capabilities. Consequently, a subsidization scheme would require a complex set of contracts among service providers for revenue sharing: an undesirable proposition. Secondly, premium services (say, a color

Proceedings of the Fifth IEEE Workshop on Mobile Computing Systems & Applications (WMCSA 2003)
0-7695-1995-4/03 $ 17.00 © 2003 IEEE

IEEE COMPUTER SOCIETY

printing service) may well be too expensive to be cross-subsidized. Finally, ubiquitous computing services might be offered in places such as a road-side kiosk where the possibility of cross-subsidization is low.

Ad-based Revenue:

Within the context of the Internet, advertisement-based revenue has been a popular model. In this model, services are paid for not by users, but by advertisers who promote their goods to users of the service. However, in the Internet domain, this model did not turn out to be a viable business proposition as attested to by the changes underway. Vendors on the Internet have realized that ad-based revenue is not sustainable and are increasingly turning to charging individual customers for service use (for examples see www.theendoffree.com). Within the context of ubiquitous computing, many different advertising modes are possible, ranging from pop-up ads and tickers on public displays to location-based advertising [6]. However, drawing parallels with the Internet, we believe that it is unlikely that advertising alone will be sufficient to cover the cost of premium ubiquitous computing services.

Public Service:

Yet another possibility is for governments to create ubiquitous computing infrastructure on the lines of public radio. Indeed, in many countries ubiquitous services such as the telephone network began as publicly funded initiatives. However, while it may be the case that some elements of ubiquitous computing infrastructure are government subsidized, it is unlikely that governments will want to take on the role of ubiquitous computing service providers. Hence, it is our belief that ubiquitous computing services and environments will thrive as a result of private enterprise driven by the desire for profits.

Charging for Service Use:

Finally, revenue for ubiquitous computing services can be recovered from end-users as is the case for many other services such as mobile telephony. Such a scheme provides immense flexibility and scope for creating revenue streams and reflects the current mechanisms used in most service industries. For example, television in the UK was established as a public service paid for by the government and was subsequently enhanced to include many subscription-based channels.

All of the above schemes provide mechanisms by which investors in ubiquitous computing infrastructure can recover their costs. In our opinion, *no one scheme will emerge as the sole mechanism* that is employed, and in practice combinations of two or more of these schemes will be used to obtain suitable returns on the investment. In this paper, we focus on issues concerned with direct payment for service use – one of the techniques we identify above as being suitable for obtaining revenue in ubiquitous computing systems. The main contributions of this paper are:

1. identification of the key requirements for payment systems for ubiquitous environments,
2. analysis of existing payment systems and their suitability for supporting payment for ubiquitous computing services,
3. a new protocol, the Payment Session Protocol (PSP) that provides rich support for payment enabled ubiquitous computing environments, and,
4. examples of the use of PSP to support charging for service use.

The remainder of this paper is structured as follows. Section 2 presents a ubiquitous computing scenario in which services are charged for and establishes the requirements for a ubiquitous computing payment system. Section 3 analyses existing electronic payment schemes and examines their suitability for use in ubiquitous computing environments. Section 4 presents our protocol (PSP) for supporting payment in ubiquitous computing environments and section 5 describes our use of PSP to support prototype payment-enabled applications. Section 6 describes related work and section 7 presents our conclusions and discusses future work.

2. Design Requirements for Ubiquitous computing Payment Systems.

So, what does it mean to charge for ubiquitous computing services? Do ubiquitous computing environments impose additional requirements for payment systems? Do existing electronic payments fulfill those requirements? To help answer such questions, we illustrate a typical ubiquitous computing scenario adapted from [7] and motivate the requirements for payment systems in ubiquitous computing environments.

Jane is at Gate 23 in the Pittsburgh airport, waiting for her connecting flight to Honolulu. She has edited documents on her laptop and would like to print a copy and use her wireless connection to e-mail them. The pervasive computing system on her laptop has already discovered the services available in the airport. The printer application prompts Jane to select her choice from the list of discovered services, displaying details of the services and their associated costs. The email client however selects the lowest priced service automatically and mails the documents. In both instances, the wallet on Jane's laptop, which Jane has configured to pay automatically, pays for the services. After collecting her printed copies, she heads towards a 'triptik' service, where she can obtain a map of the route from Honolulu airport to her hotel. Upon payment request, Jane enters her remote wallet id as her laptop is powered down. She

IEEE COMPUTER SOCIETY

is also issued an e-tag by the 'triptik' service that identifies her to other services in the airport. Poring over the map, she strolls across to a café where she can watch clips of the previous day's super bowl. The scanner recognizes her e-tag and the payment requests for her coffee and the football clips are directed to her remote wallet. Putting down her coffee cup, Jane realizes that her flight departs in the next 10 minutes, invalidates her e-tag and heads towards the gate.

From the above sketch, we see that ubiquitous computing interactions are typically spontaneous and short-lived, often initiated by mobile users and involving numerous trusted and untrusted geographically dispersed services. In the following sections, we derive the design requirements for payment systems for ubiquitous computing environments in light of these characteristics.

Spontaneity

Kindberg and Fox [8] argue that spontaneity is an inherent and desirable characteristic of ubiquitous interactions. They identify the volatility principle that states that ubiquitous computing systems should be designed "on the assumption that the set of participating users, hardware and software is highly dynamic and highly unpredictable". In terms of payment systems, this means that it is highly unlikely that individuals will enter into long-standing relationships with all of the different service providers they are likely to encounter. We believe that ubiquitous computing payment systems should thus allow users to adopt a pay-as-you-go model for service use.

Efficiency

A lot of efficiency requirements of ubiquitous computing payment systems flow out of the trust relation between users and service providers. More clearly, in situations where the user and the provider do not have an established trust relationship, frequent low-value payments for incremental service provision are more likely. When many small payments are involved, it is important that the payment process is lightweight and efficient; characterized by low communication and computation costs coupled with low financial overheads. The computational and communication considerations are especially important when we consider the resource limitations typically associated with ubiquitous computing devices. Finally, many existing payment schemes incur a significant financial overhead for each transaction (for example, credit card companies' processing fee is typically of the order of 50 cents for each transaction). Such an overhead is clearly unacceptable in ubiquitous computing environments where frequent low-value transactions are involved.

Security

Clearly, security is an important consideration in any payment system and adequate safeguards have to be taken against frauds such as theft, counterfeit money, and payment evasion. Ubiquitous computing environments that are created anywhere and by anybody are more vulnerable to security breaches than controlled environments that can be physically secured. The issue of security also arises due to the lack of established trust relationships between end-users and service providers. For example, while users paying upfront for a service are not assured of the service, service providers charging users after delivering a service are not assured of payment. Payment systems that do not address above security concerns are not acceptable to users or service providers.

Privacy

Many payment systems like credit cards and money transfers require users to disclose personal information such as their name and account numbers as part of the transaction. One could envision ubiquitous scenarios in which users do not wish to disclose this information but need to pay for services. Furthermore, without adequate privacy protection mechanisms, payment information could be combined with other contextual information providing detailed information on users' activities. Thus, for payment systems to be acceptable to users, privacy should be addressed in the design of ubiquitous computing payment systems from the beginning, rather than as an afterthought.

Flexibility

As mentioned previously, ubiquitous systems should abide by the volatility principle and not assume any specific configuration of networks, devices, and users. While we generally consider ubiquitous environments to be rich in device and network resources, we can identify two important special cases, disconnected operation and device unavailability.

Disconnected operation is a mode of operation that enables clients to continue accessing services during temporary failures of a shared data repository [9] or a network connection. In the future, when ubiquitous computing environments proliferate, they are likely to span areas that vary widely in network characteristics. Hence, payment systems should not be based on assumptions of continuous connectivity.

On the issue of device unavailability, user interactions with ubiquitous computing environments may or may not involve a personal device such as PDA or a smart phone. However, users still need to pay for the service use. It is therefore important to not make payment rely on the use of users' personal devices. This issue was highlighted in the above airport scenario when Jane uses her remote wallet to pay for the 'triptik' service.

IEEE COMPUTER SOCIETY

Usability

A ubiquitous computing payment system is likely to be involved in many transactions during the course of a typical day and it is crucial that special attention be paid to the usability aspect. For instance, users will expect a level of service comparable to existing credit cards and bank accounts. Issues such as trust, liability, accounting and insurance must be adequately addressed for users to accept the system. Payment systems face new and difficult research challenges unique to ubiquitous computing environments. For example, Mark Weiser espoused the notion of calm technology [10] where computing disappears into users' subconscious. However, most users would not accept a system that allows arbitrary financial transactions to take place without their involvement. At the same time, it is not practical for user involvement in all transactions when transactions are typically of low value. Hence, designers of payment systems are faced with balancing the contradictory needs of calmness and user involvement.

Deployability

In order to be successful, any ubiquitous computing payment system must be capable of being deployed on a wide scale. In practical terms, this will require support for both new and existing services. It is a common experience that if new concepts such as payment necessitate significant changes to existing applications, they are unlikely to be adopted by users at large. Hence, payment systems should be engineered so as to require minimal or no changes to the existing code base.

The above requirements highlight some of the key challenges in designing a payment system for ubiquitous computing environments.

3. Existing Payment Schemes.

Existing electronic payment systems can be broadly classified into two categories: macropayment systems and micropayment systems. Macropayment systems, the most widely used form of payment, are exemplified by credit cards and electronic money transfers. Macropayment systems are generally designed to support transactions involving medium-high valued payments. In contrast, micropayment systems are generally characterized by low-value transactions. Examples of micropayment systems include Millicent [11] and MicroMint [12]. In the following sections, we provide an overview of these two classes of payment system and analyze their suitability for payment in ubiquitous computing.

3.1. Macropayments

Macropayment schemes are generally designed to support medium-large payments of the order of $5 or more [11]. Typical examples of macropayment systems include credit cards, subscriptions, and bank checks. Since the value of macropayment transactions is generally high, such systems are characterized by a requirement for strong security measures. For example, the theft of someone's credit card number or a bank account number could result in the loss of a significant amount of money by users or merchants. Hence, security measures such as strong encryption schemes and centralized authorization brokers are typically an integral part of macropayment systems. As a result of this desire for strong security together with associated accounting rigour and billing requirements, macropayment systems typically incur high overheads in terms of computational resources, set up costs, and processing fees. Such overheads result in significant financial costs being associated with each transaction in a macropayment scheme. For example, a typical credit card transaction incurs a processing cost of 50 cents or more. It is important to note that although Macropayments are occasionally used for small payments, they are designed for high-value transactions, for which these overheads represent a small percentage of the total transaction.

Examining today's e-commerce systems we are able to identify three frequently used payment mechanisms that can be classified as macropayment based.

Subscriptions: Subscriptions represent the simplest model for e-commerce. In this scheme, customers create accounts with vendors and are billed periodically. During service initiation, users identify themselves as legitimate users by providing proper credentials. Vendors add service costs to the user account and bill the user at regular intervals.

Credit Cards: In contrast to subscriptions, credit card schemes allow users to pay as they go without having to establish accounts with every vendor. Instead, both the user and the vendor enter into long-term agreements with a centralized broker, e.g. a bank or a credit card company. The broker generally represents a trusted entity who is contacted on every transaction for authentication and payment authorization. Brokers typically impose a processing fee for providing their services.

Aggregation: Aggregation is a form of subscription wherein transaction costs are accumulated at the vendor until they exceed some threshold. At that point, the aggregation service deducts the accumulated amount in a single transaction from the user's credit card or bank account. Thus, aggregation amortizes billing charges over a sequence of less expensive transactions. Aggregation services are provided by vendors themselves or by third-party services such as Pay-Pal [13].

3.2. Micropayments

In contrast to macropayments, micropayments are lightweight payment schemes designed to support low-

IEEE COMPUTER SOCIETY

valued transactions of the order of a few cents. In this model, users obtain digital cash from an authorized broker in bulk using conventional payment methods such as cash or credit cards. This digital cash can be used to pay for services directly or can be exchanged for vendor specific digital cash that can be used to pay for a service from a particular vendor. Thus, micropayment schemes can operate as a hierarchy of issuers and provide for lightweight validation of cash without the need to have cash validated always by a central authority. Micropayments were conceived to be used to pay for low-valued items such as web pages, ringtones or small multimedia clips.

As a result of their focus on low-value transactions, micropayment systems have lower security requirements than their macropayment counterparts and thus are characterized by lightweight encryption techniques. More specifically, encryption techniques need to be only strong enough to make the costs of breaking them greater than the potential monetary benefits. Moreover, the loss of a few Micropayments is akin to losing small change in a vending machine and generally considered less acute than losing a credit card number. Hence, lightweight encryption techniques are sufficient for micropayment-based payment systems. Such encryption schemes decrease the latency of a payment transaction and have significantly lower computational costs. Furthermore, Micropayments typically do not incur overheads such as account setup.

3.3. Analysis

Given the above classification of payment schemes we can begin to explore the relative merits of macropayment and micropayment schemes for use in ubiquitous computing environments.

Considering spontaneity first, we note that in ubiquitous computing payment systems users should be able to pay for services without having to establish a long-term relationship with service providers: it is impractical for users to have the knowledge of services at different locations ahead of time. Such spontaneous interactions are impossible in systems that require advance account set-up.

In terms of efficiency, micropayment systems are designed to be more efficient than macropayments in terms of low financial and computational costs and lower payment latencies. Almost all of the macropayments incur high financial processing costs and high computational costs by virtue of using strong encryption mechanisms. High payment latencies are also associated with services such as MSN Passport and PayPal that require connection to a central broker to authorize transactions.

As regards privacy, macropayment schemes such as credit cards and third party services such as MSN Passport require users to disclose their personal information, allowing different vendors to collude and infer the buying patterns of users. This clearly amounts to a violation of users' privacy. In contrast, the digital cash used in Micropayments does not contain any personal information, preventing vendors from colluding to profile users. However, brokers who dispense the broker scrip to users and exchange the broker scrip for cash with vendors are still able to mine information about users. However, in practice, brokers are likely to be few in number and generally more trustworthy than vendors (brokers would be akin to banks) and hence Micropayments generally offer more privacy than Macropayments.

Finally, macropayments such as credit cards and aggregation schemes are based on strong connectivity to brokers for payment authentication and authorization. In contrast, micropayment systems often utilize authorization hierarchy and avoid contacting a central authority for every transaction. The relative merits of macro and micropayments in terms of requirements for ubiquitous computing payment systems are summarized in table 1.

Table 1. Comparison of macropayment and micropayment systems

System/Property	Macropayments	Micropayments
Spontaneity	Not supported	Supported
Encryption costs	High	Low
Broker interaction	Frequent	Infrequent
Financial costs	High	Low
Disconnected operation	Not supported	Supported
Payment Latency	High	Low
Anonymity	Low	High
Security	Medium-High	Low-Medium

In general, macropayment schemes such as credit cards, third-party aggregation services such as PayPal and other subscription schemes fail to meet one or more of our requirements of spontaneity, low financial processing costs, low latencies, anonymity and disconnected operation. Therefore, we believe that future ubiquitous computing payment systems will be based on micropayment schemes that have the potential to satisfy the design requirements we have identified. In the following sections, we describe a prototype micropayment based solution designed to support payments in ubiquitous computing environments.

4. Payment Session Protocol.

Consider a typical commercial transaction in the real world such as buying a coffee from a café. After the user walks into the café, the price of the required item is found out either by looking at the menu or asking the shop keeper. In some cases the price may be open to

IEEE COMPUTER SOCIETY

negotiation or reduction through the use of, for example, discount vouchers. Once the price is agreed, cash is paid or credit card details disclosed so that the appropriate amount is billed upon service use. If the merchandise is to be collected at a later time, the merchant might even issue a receipt or a token that helps in identifying the user while claiming the merchandise. This example illustrates that the actual payment mechanism (cash or credit card) is only one component of a rich set of interactions that are required to conduct even the simplest of commercial transactions. Similarly, for ubiquitous computing environments where services are charged, the provision of payment infrastructure requires flexible integration with other ancillary protocols for service discovery and interaction. More specifically, we can identify five key stages in a typical commercial transaction or service use as shown in figure 1.

1. Obtaining means of payment – *this stage involves obtaining the appropriate means of paying for the service, e.g. obtaining digital cash in an appropriate currency.*
2. Service discovery and selection – *in this stage the client discovers available services and selects based on a range of factors including price.*
3. Payment negotiation – *once an appropriate service has been selected the client can negotiate payment related parameters such as payment method and mechanism for authentication.*
4. Service use – *clients use the service in question, possibly making on-going payments during this phase.*
5. Termination - *this is an extension of stage 3, in which at the end of service use clients reclaim any unspent money or obtain a proof of payment and service use.*

Figure 1. Stages of service use in payment-enabled systems

For any given transaction not all of these stages will be applicable and the order may differ from other transactions. For example, client applications may do service discovery and price negotiation ahead of obtaining the appropriate currency and use the service subsequently. Hence, these stages should simply be viewed as a set of tasks that must be supported in order to provide comprehensive support for a broad range of commercial transactions.

In the following sections, we present a new protocol termed the Payment Session Protocol (PSP) that provides support for payment-related interactions between clients and servers. It should be noted that PSP does not address issues such as service discovery since numerous protocols already exist to support these issues. PSP reflects commercial transactions in the real world and is based on the exchange of contracts. These contracts describe the non-functional characteristics of services such as QoS, cost and terms and conditions of use and are created to provide service level agreements between clients and servers. PSP is designed to operate in conjunction with a micropayment-based payment solution. In our current architecture we have focused on interoperating with the Millicent micropayment system [11].

It should be noted that one critical component of any payment system is maintaining association between payment and service use. In a secure environment in which clients and servers are always authenticated prior to service use this is a straightforward problem since the identities of the parties involved are know and payment can be associated with specific clients, servers and instances of service use. However, in the more general case, maintaining an association between service use and payment is a challenging problem. For example, if a client pays for access to a service using one payment-specific protocol and then accesses the service using a second service-specific protocol, the service provider needs to ensure that only clients who have paid have their requests honored. In ubiquitous environments we believe that it is unreasonable to assume a secure environment or a single mechanism for identifying clients. Hence, we adopt an approach in which the mechanism for maintaining the association between service use and payment is negotiated as part of the contract exchange process. For example, a contract may specify either that service invocations are accompanied by specific session identifier or are acceptable only from a specified client address. Where services are of a high-value more rigorous means of associating service use and payment (such as using a secure channel for both payment and service use) must be specified in the contract. In short, no one method of associating payment and the service use suffices and the particular method depends on the service and the corresponding terms of service use.

4.1. Underlying Payment Scheme: Millicent

PSP could work in conjunction with multiple payment schemes. For the purposes of our research however we have focused on the use of the Millicent protocol as an underlying payment protocol. Millicent is based on two kinds of scrip: 'broker scrip' and 'vendor scrip' that are specific to, and validated by, brokers and vendors respectively. When a customer makes a purchase with a scrip, the cost of the purchase is deducted from the scrip's value and a new scrip (with the new value) is returned as change. When the customer has completed a

IEEE COMPUTER SOCIETY

series of transactions, they can "cash in" the remaining value of the scrip.

Brokers, serving as accounting intermediaries, buy and sell 'broker scrip' and 'vendor scrip' as a service to customers and vendors. Users buy 'broker scrip' when they do not have prior knowledge of services they will use. In such instances, 'broker scrip' serves as the common currency, accepted by all vendors. When users utilize the services of a vendor, they initially tender the broker scrip. The vendor after validating the 'broker scrip' with the broker returns 'vendor scrip' to the user, that is subsequently used for paying that vendor. It might be noted that the vendor contacts the broker only once for validating the broker scrip and thereafter validates the 'vendor scrip' locally. Alternatively, if users have prior knowledge of the vendors they will interact with, they can buy the corresponding vendor scrip from brokers. In such instances, vendors need not contact brokers as the vendor scrip is validated locally. Thus, Millicent, by localizing the validation of the vendor scrip, survives network disconnections between a Ubiquitous computing environment and the outside world.

The sequence of interactions among the three entities is captured in the following diagram.

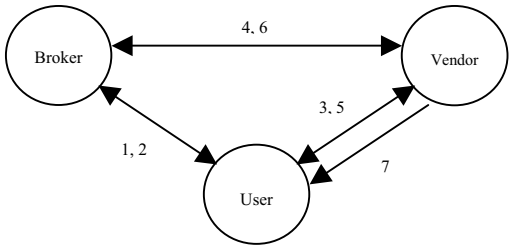

Figure 2. The sequence of interactions among brokers, users and vendors

In step 1 and 2, users request and obtain 'broker scrip' or the 'vendor scrip', using some form of macro-payment such as credit cards. If the broker scrip is bought from the broker, in step 3, the user connects to the vendor over a secure channel and tends the broker-scrip to the vendor. The vendor then validates the broker scrip by contacting the broker (4). In step 5, the vendor returns its own scrip (the vendor scrip), a certificate and a session key that is used to encrypt any further sensitive communication between the user and the vendor. It should again be noted that the vendor talks to the broker for validating the 'broker scrip' only once. Later on, when the user pays with the vendor scrip, the vendor itself validates the scrips. After the service use is complete, the vendor in step 6 translates any unspent vendor scrip into broker scrip and returns it to the user in step 7. The user can then use that broker scrip to pay other vendors or exchange it with the broker for other forms of payment.

Millicent only defines the vendor scrip format and its associated authentication mechanism. A 'vendor scrip' consists of the *'Scrip body'* and its certificate. The format of the 'Scrip Body' is shown in figure 3.

Vendor	Value	ID#	Cust ID#	Expiration Time	Other Data

Figure 3. Format of the Millicent vendor scrip

The certificate is the result of hashing the scrip body with 'Master Scrip Secret' (*MSS*), which is known only to the vendor, i.e.

Certificate = hash (*Scrip Body*, *MSS*);
Vendor Scrip (S_V) = <*Scrip Body*, *Certificate*>

Using the certificate contained in the scrip, the vendor ensures that the scrip has not been tampered with. He further checks that the scrip has not been used by comparing it against the database of spent scrips. If valid, the vendor services the request and if any change is left from the vendor scrip, returns a new scrip S'_V. S'_V is then used for further payments.

It should be noted that Millicent is one of a number of micropayment protocols and a detailed comparison of different micropayment schemes can be found in [15].

4.2 PSP Contracts

Contracts in PSP are documents that specify the terms and conditions of service provision and use. They have obvious equivalents in the real world such as the contracts exchanged during real-estate transactions. The general format of a PSP contract is shown in figure 3.

```
Contract          ::=  Date, ContractId,
                       ( [ ServiceDesc ]+ |
                       [ JobDescription ]? ),
                       [ signature ]?,
                       [ version history ]*;
ContractId        ::=  uri;
ServiceDesc       ::=  [ ServiceEntry ]+,
                       Association,
                       [ terms and conditions ]?;
ServiceEntry      ::=  Description, ServiceId,
                       Cost, [ QoS ]?;
JobDescription    ::=  ServiceId, [ Argument ]*,
                       [ Payment ]*,
                       [ sessionId |
                       application specific
                       attributes ]?;
ServiceId         ::=  uri
Argument          ::=  string, "=", string;
Payment           ::=  Currency, PaymentMethod,
                       [ payment | wallet uri ]+;
Association       ::=  "PSP", AssociationData |
                       "InBand",
                       [ "SecureChannel" ]?,
                       AssociationData;
```


IEEE COMPUTER SOCIETY

```
AssociationData::=   "SessionId", sessionId |
                     "ApplicationAttributes",
                     attribute specification;
Cost           ::=   Currency,
                     PaymentMethod,
                     Price;
Currency       ::=   "MillicentScrip" |
                     string;
PaymentMethod  ::=   "Prepay" |
                     "Postpay" |
                     "OnDemand";
Price          ::=    price specification;
```

Figure 4. PSP contract definition

Contracts are composed of three basic components: information about the contract itself (identifiers, version history, signatures etc.), a description of one or more services offered by the service provider or a description of one or more services that the client wishes to pay for (akin to a purchase order). Service descriptions include service details and associated non-functional parameters such as cost, QoS and terms and conditions of use. In addition, service descriptions specify how service use is to be correlated with payment, as discussed previously.

It should however be noted that contracts are not as heavyweight as they might sound and are a vehicle to explore and accommodate the requirements of different transactions. In a simplistic scheme, a contract might include just the description and the cost of the service and the contract negotiation might just involve paying for the service. We have not at present, defined all aspects of contracts in detail: to do so would be a major undertaking. Witness for example, the complexity in developing a generic mechanism for specifying privacy policies in a machine-readable format [16]. However, Figure 4 offers an insight into the likely content of contracts and provides sufficient detail to motivate the remainder of this paper.

4.3 PSP Commands

PSP is used to support the exchange of contracts and payment between clients and servers. In other words, PSP provides a framework for integrating payment into applications. The protocol supports the basic tasks of obtaining information on available services, negotiating the terms and conditions of service provision and use (including the cost), paying for service use and management of service use agreements.

The specific operations supported in PSP are shown in Table 2 and described in the remainder of this section.

Table 2. PSP operations

Operation	**Return Results**
GetServiceDesc (query)	Draft Contract(s)
Submit (contract)	Contract (possibly signed)
Invalidate (contractId)	Status
Pay (contractId, payment, args)	Status
GetPayment (contracted, args)	Status or Payment
GetStatus (options)	Status

GetServiceDesc (query)

Clients wishing to use a payment enabled service can obtain detailed information about the characteristics of the service by invoking the `'GetSeviceDesc'` method on the service's PSP endpoint. This information is returned as a draft contract, optionally signed by the server as described in section 4.2. In many cases clients will already be aware of the characteristics of the service they wish to use and may already have cached a draft contract. In such cases this operation will not be used.

Submit (contract)

Having obtained a draft contract for a service, a client can decide on the specific details of its service use, complete a job description associated with the contract and submit a (optionally signed) version to the server. If the client has accepted the terms and conditions of the server's draft contract, the `'Submit'` operation creates a binding agreement between the two parties. However, the client can also choose to modify parts of the contract and submit this to the server as part of a negotiation phase. Such negotiation could involve repeated exchanges of contracts between clients and servers. However, in general, we believe that clients will typically accept the draft contracts proposed by service providers and are unlikely to become involved in lengthy negotiations with the server.

Invalidate (contractId)

Subject to the terms and conditions of a contract, a client or server can choose to terminate a previously agreed contract by invoking the `'Invalidate'` operation. This termination could reflect the satisfactory conclusion of a transaction or, for example, a client's desire to no longer use the service offered.

Pay (contractId, payment, args)

Once a contract has been agreed clients can use the Pay operation to send payment for a service to the service provider. This payment may take the form of Millicent scrip or a URI through which the server can obtain payment. This enables, for example, third-party payment for service use. The arguments associated with the `'Pay'` operation are contract/service specific and are used to, for example, associate the payment with a specific use of the service.

IEEE COMPUTER SOCIETY

GetPayment (contracted, args)

In cases where clients have presented a URI to the server in lieu of payment the service provider invokes the `'GetPayment'` operation on the endpoint specified by this URI. Once again the arguments are contract/service specific and can be used to ensure that the endpoint from whom payment is being requested is able to verify that the request is bona fide. Besides, this operation should confirm to transaction semantics for security and consistency.

GetStatus (contractId, options)

`'GetStatus'` is used for general management functions such as enabling clients to enquire about the status of an agreement or associated session. This operation can also be used to obtain additional information from the service provider such as a receipt for service use or a copy of the contract between the client and server.

In the general case, clients and servers pass through the stages of service discovery, negotiation, payment, service use and termination using the operations described above. However, there are several optimizations that can be carried out that help to reduce the overhead associated with PSP. Specifically:

(i) Clients can cache draft contracts or obtain contracts as part of their general service discovery mechanisms – hence removing the requirement to call `'GetServiceDesc'` on service providers.

(ii) Clients who hold draft contracts for a service can send payment together with their job request – removing the need for a separate interaction for payment.

(iii) Completion of service use can signal termination of the agreement and hence `'Invalidate'` messages may not be required.

A subset of PSP can also be implemented as an extension to existing application protocols such as HTTP or SMTP. Hence, in an optimal scenario, clients could simply send a URI for a contract and associated job request together with payment as part of the extended application protocol. In this paper we do not specify how PSP should be engineered. Indeed, it maybe that different environments would wish to support PSP using underlying transport protocols that are supported within the specific environment.

5. Integrating Payment into Applications.

Critical to the success of PSP is the ease with which it can be integrated into new and existing applications. In this section we explore the practical implications of including PSP in a ubiquitous computing environment by considering how PSP can be integrated into our existing system to control access to a public wireless network.

As part of our on-going work into providing public access to ubiquitous computing infrastructure we have deployed a wireless city-wide public access network based on 802.11. This network runs IPv6 and public access is managed using a custom public access control system developed in-house [17]. This system is based on the concept of packet marking and filtering. More precisely, each IPv6 packet sent from an authenticated client terminal contains within the IPv6 extension header an access control token (ACT), which is then verified by the router that controls network access (AR). Based on the presence and validity of the ACT, access to the rest of the network is either granted or denied. The ACT is generated and distributed to the client terminal and the access control router by the Authentication Server (AS) after the user is authenticated. A detailed description of our access control mechanism and the secure token distribution protocol can be found in [17].

Currently we authenticate clients (and hence control access to the network) by requiring users to log-in to the system using a user name and password. Successful authentication results in an ACT being distributed to the client. To protect the user name, password and ACT a secure channel is used for the log-in process. In order to support public access to the network based on payment rather than credentials the following changes would be required. Firstly, the AS would need to be extended to support PSP and to provide a suitable PSP end-point through which clients could interact with the AS. The second step would be to formulate an appropriate contract for network access. Such a contract would need to describe the service and associated terms and conditions and would clearly have much in common with existing service agreements between ISPs and users. For the remainder of this example we assume that the contract states that network connectivity costs 10c per minute with payment being required for each minute in advance. Since the service provider does not trust the client they are requesting payment prior to providing the service. Of course, since the client does not, in turn, trust the service provider they are unlikely to wish to pay for more than one minute in advance; hence payment will need to be collected in parallel with service use.

Client devices entering our network domain would first request the draft contract for service use from the AS's PSP end-point. Assuming they agree to the terms and conditions they would complete the necessary details in the contract and send a copy back to the AS. Since the contract requires payment in advance the client sends a URI and an identifier as part of the job request component of the contract that is returned to the AS. The AS can then invoke the `GetPayment` operation on this URI and obtain Millicent scrip sufficient to cover the first minute of operation. The AS can then issue the client with an appropriate ACT and the client can access the network.

IEEE COMPUTER SOCIETY

For each minute the client accesses the network the AS can request additional payment from the URI. In order to protect the client from fraud the AS is required to present identifying information when making this request. The nature of this identifying information can be specified as part of the contract and the level of security can be selected to match the perceived risks and costs. If the client does not send data for a specified period (or payment cannot be obtained) the server can terminate the agreement and invalidate the ACT at the AR.

It is important to note that since we need to protect the ACT from eavesdroppers the ACT must be sent to the client over a secure channel, just as in the existing scheme.

6. Related Work.

To the best of our knowledge, we are the first to explore the concept of payments in ubiquitous environments. The concept of charging for services has however been explored in considerable depth in the context of the Internet [18]. Of late, the economics of peer-peer systems has been receiving a lot of attention. In particular, P2P systems such as Mojonation are attempting to create a file-sharing economy of agents, servers, and search engines in which senders and receivers can agree on prices for each transaction [19][20]. Grid computing is another area where payment issues are beginning to receive attention [21].

The applicability of micropayments for information goods such as web pages has been explored by both researchers [11][12], and commercial enterprises. However, while proposals such as Millicent and NetBill [22] reached the trials stage, other proposals such as MicroMint remained on paper. The World Wide Web consortium formed a working group on micropayments and defined html extensions to include payment information. After steering an initial implementation of the specification, the working group wound up its activity in 1999 [23].

7. Discussion and Future Work.

In this paper, we have drawn attention to the importance of economic considerations for the proliferation of ubiquitous computing environments. More specifically, the deployment of ubiquitous applications and environments involves huge investments, and hence there must be a financial underpinning that encourages such investment. To this end, we discussed different ways of recouping investments in ubiquitous computing environments: cross-subsidization, ad-based revenue, public investment and charging for service use. Subsequently, we motivated the need for charging for ubiquitous computing services and identified the design requirements of payment systems for ubiquitous computing environments. By analysing existing electronic payment schemes, namely macropayment and micropayment systems, we concluded that micropayment protocols provide a firm basis for payment in ubiquitous computing systems. We presented our new protocol, the Payment Session Protocol (PSP) that represents an initial step towards providing a framework for enabling payment in ubiquitous computing systems. We also presented an example of a payment-enabled prototype application that provides insights into the mechanics of ubiquitous computing payment systems and serves as a proof-of-concept for our Payment Session Protocol.

While our initial impressions and efforts look very promising, there is more ground to cover before payment-enabled ubiquitous computing services can be widely deployed. In addition to creating a payment infrastructure of clients, service providers and brokers, further research is required in two key areas: pricing structure and user acceptance. Within the context of pricing structure, there is a need to investigate techniques for expressing the richness and complexity of pricing schemes that we anticipate occurring in ubiquitous computing environments (witness the proliferation of pricing schemes in the field of mobile telephony). In terms of user acceptance, balancing the need to provide users with information on financial transactions with the desire for calm technology will clearly present a significant challenge to researchers.

Finally, we note that PSP offers the potential to support a wide range of features such as privacy policies, liability statements and terms and conditions of service. These properties will become crucial in ubiquitous computing environments where users are expected to interact with a wide variety of trusted and untrusted services. Walking into a new ubiquitous computing environment should not involve the user in clicking "accept" on dozens of service agreements and yet such agreements will have to be in place to conform to legal and ethical norms of service use. In short, we foresee a significant series of research challenges arising in this area for which PSP provides a starting point for further exploration.

8. References.

1. M. Weiser, “The Computer for the Twenty-First Century”, *Scientific American*, September 1991, pp. 94-100.
2. R. Want, A. Hopper, V. Falcao, and J. Gibbons, “The active badge location system”, *ACM Transactions on Information Systems*, vol. 10, Jan. 1992, pp. 91-102.
3. A. Fox, B. Johanson, P. Hanrahan, and T. Winograd, “Integrating information appliances into an interactive workspace”, *IEEE Computer Graphics and Applications*, vol.20, no.3, May-June 2000.

IEEE COMPUTER SOCIETY

4. M. Román, C.K. Hess, R. Cerqueira, A. Ranganathan, R.H. Campbell, and K. Nahrstedt, "Gaia: A Middleware Infrastructure to Enable Active Spaces", *IEEE Pervasive Computing*, Oct-Dec 2002, pp. 74-83.
5. N. Davies and H.W. Gellersen, "Beyond Prototypes: Challenges in Deploying Ubiquitous Systems. System Software for Ubiquitous Computing", *IEEE Pervasive Computing: Mobile and Ubiquitous Systems*, Vol. 1, No. 1, January - March 2002.
6. A. Ranganathan and R.H. Campbell, "Advertising in Pervasive Computing Environment", *ACM International Workshop on Mobile Commerce*, Atlanta, Georgia, September 28, 2002, pp 10-14.
7. M. Satyanarayanan, "Pervasive Computing: Vision and Challenges", *IEEE PCM*, August 2001, pp. 10-17.
8. T. Kindberg and A. Fox, "System Software for Ubiquitous Computing", *IEEE Pervasive Computing: Mobile and Ubiquitous Systems*, Vol. 1, No. 1, 2002.
9. J.J. Kistler and M. Satyanarayanan, "Disconnected Operation in the Coda File System", *ACM Symposium on Operating Systems Principles*, 1992.
10. M. Weiser and J.S. Brown, "The coming age of calm technology", *PowerGrid Journal*, Version 1.01, July 1996.
11. S. Glassman, M. Manasse, M. Abadi, P. Gauthier and P. Sobalvarro, "The Millicent Protocol for Inexpensive Electronic Commerce", *Proc. 4th Intl. World Wide Web Conference*, Boston, MA, Dec. 1995.
12. R. Rivest and A. Shamir, "Pay Word and Micro Mint: Two simple micropayment schemes", *Security Protocols Workshop*, 1996.
13. http://www.paypal.com.
14. http://www.passport.net.
15. C. Ellis, "Evaluation of Micropayment Schemes", *HP Labs Technical Report*, HPL-97-14.
16. G. Myles, A. Friday, and N. Davies, "Preserving Privacy in Environments with Location Based Applications", *IEEE Pervasive Computing: Mobile and Ubiquitous Systems*, Vol.2, No.1, Jan-March 2003.
17. S. Schmid, J. Finney, M. Wu, A. Friday, A. Scott, and D. Shepherd, "An Access Control Architecture for Microcellular Wireless IPv6 Networks", *Proceedings of 26th Annual IEEE Conference on Local Computer Networks (LCN 2001)*, November 2001.
18. Internet Economics Workshop, http://www.press.umich.edu/jep/econTOC.html.
19. Get Your Music Mojo Working, Wired Magazine, http://www.wired.com/news/technology/0,1282,37892,00.html.
20. http://www.mojonation.net.
21. A. Barmouta and R. Buyya: GridBank, "A Grid Accounting Services Architecture (GASA) for Distributed Systems Sharing and Integration", www.cs.mu.oz.au/~raj/grids/papers/gridbank.pdf.
22. M. Sirbu and J.D.Tygar, "NetBill: An Internet Commerce System Optimized for Network Delivered Services", *IEEE Personal Communications*, August 1995.
23. T. Michel, "Common Markup for micropayment per-fee-links", *W3C Working Draft*, 25 August 1999, http://www.w3.org/TR/WD Micropayment-Markup/.
24. D. Chaum, "Blind signatures for untraceable payments", *Advances in Cryptology Proceedings*, *Crypto 82*, 1982, pp. 199-203.
25. Schneier, B., *Applied Cryptography: Protocols, Algorithms, and Source Code*, John Wiley and Sons, Inc., New York, NY, USA, 1994.
26. A. Odlyzko, "The bumpy road of electronic commerce", *WebNet 96 - World Conf. Web Soc. Proc., H. Maurer, ed.,* AACE, 1996, pp. 378-389.
27. P. C. Fishburn, A. M. Odlyzko, and R. C. Siders, "Fixed fee versus unit pricing for information goods: competition, equilibria, and price wars", *First Monday*, vol. 2, no. 7, July 1997.

IEEE COMPUTER SOCIETY