InterPay: Managing Multiple Payment Mechanisms in Digital Libraries

Steve B. Cousins Steven P. Ketchpel Andreas Paepcke
Hector Garcia-Molina Scott W. Hassan Martin Röscheisen
Stanford University
{cousins,ketchpel,paepcke,hector,hassan,rmr}@cs.stanford.edu

March 20, 1995

Abstract

We propose an architecture called InterPay for managing financial interactions with for-pay digital library services. The approach accommodates multiple payment mechanisms, interaction models, and charging policies. Key components of our model are *payment agents* and *payment capabilities* that encapsulate payment policy and the details of payment on behalf of the user. *Collection agents* and *collection capabilities* provide similar encapsulation for the service provider. The architecture supports interactions ranging from individual users directly interacting with the service provider to institutional users accessing information brokers via a corporate library. We also describe a prototype system that implements the InterPay architecture, allowing access to real services under varying payment policies.

1 Introduction

In an ideal world, access to information and services would be free. However, collecting and providing access to information and services is expensive and these costs must be recouped. To permit this recovery, a number of electronic payment methods have been developed, including First Virtual [9], NetCheque [8], DigiCash [3], and many others are on the way. These mechanisms have different properties, and it is unlikely that any single one will be sufficient for universal use. Therefore, clients and services need to handle a variety of protocols.

Users of a digital library will access it in many ways, involving different payment strategies. For example, sometimes a user will directly contact an information provider, but at other times, there may be several layers of intermediaries: a university library, a for-pay information broker, and/or a database company that provides access to multiple independent databases. The different parties involved in an interaction might require different payment types or might allow new payment types, such as the corporate customer accounts used to access Knight-Ridder's Dialog Information Service [7]. These parties might also have different charging policies, for instance charging a fixed monthly rate, charging for connect time, or charging per document retrieved. Our goal is to develop an architecture that can effectively support this wide variety of charging policies and payment protocols.

Another difficult aspect of financial transactions is authorization. On the one hand, the user needs to be protected from fraudulent charges and needs to be informed of mounting transaction costs. On the other hand, seeking explicit authorization for all transactions may be too disruptive when there are a large number of "micro-transactions." In addition, there may be more than a single person involved in the authorization, e.g., the end-user, his superior, or a broker working for the end-user or his organization. In our InterPay architecture, it is possible to separate the authorization policies from the rest of the information processing activities and to provide convenient user interfaces to the required people.

Each of these aspects, diverse payment mechanisms, diverse charging models, and diverse user interactions, is a potential source of complexity for library clients and services. If they are implemented in an *ad hoc* way, it will be difficult to add new functionality as required. The InterPay architecture we propose

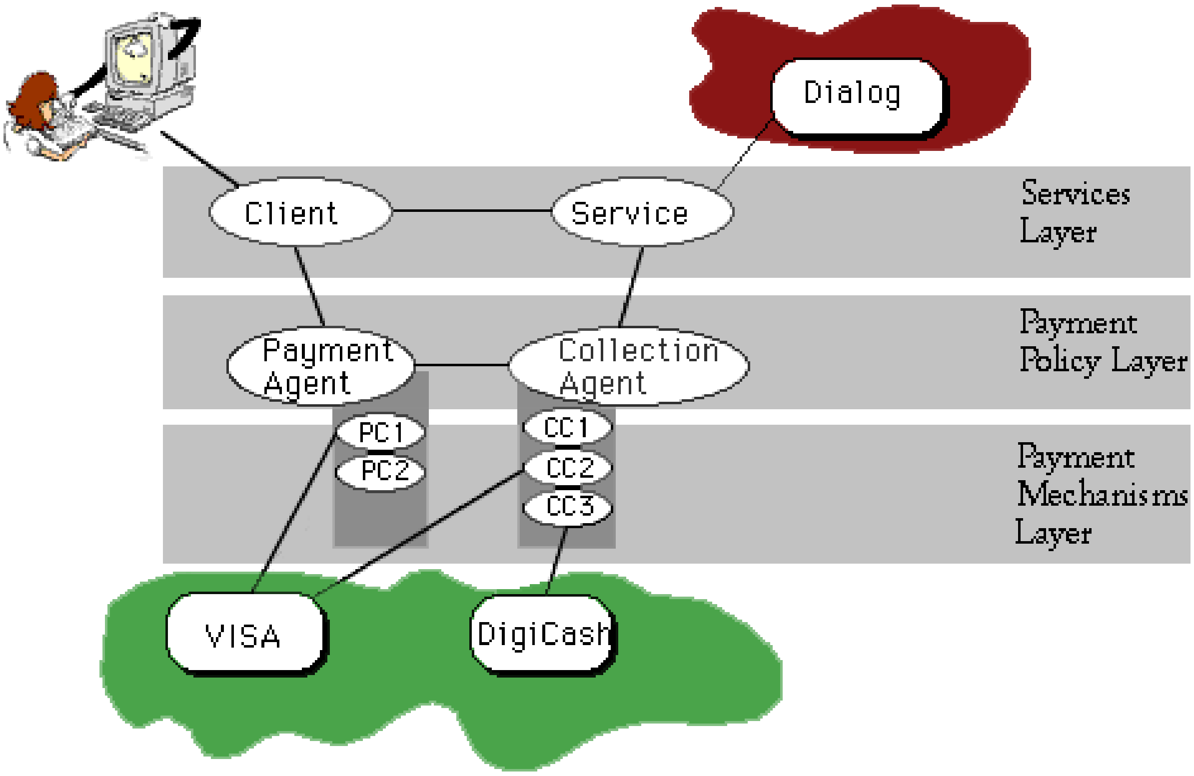

Figure 1: The InterPay Architecture. (PC_i denotes a "payment capability" of the Payment Agent, and CC_i denotes a "collection capability" of the Collection Agent.)

here provides a principled way of managing this complexity by creating modular agents and capabilities. It provides the "glue" that allows heterogeneous payment mechanisms and policies to be used interchangeably.

In the next section, we introduce the major components of the InterPay architecture. In Section 3, we will present a detailed example, showing how search and payment processes are treated in InterPay, and showing the specific message exchanges that take place. In Section 4, we will introduce some possible variations and other scenarios to show how our approach generalizes. A preliminary version of the InterPay architecture has been implemented, allowing charges to be made and authorized as actual information sources are accessed. In Section 5 we give a brief overview of our working prototype, and discuss some other issues raised by our work. The appendix contains a table of messages used for communication between the major InterPay components.

2 The InterPay Architecture

Our InterPay Architecture separates the task of accessing and paying for information into three layers: *services, payment policies,* and *payment mechanisms.* These layers correspond to how payment occurs in the real world. To illustrate, consider a father shopping for a toy with his child. Once the child selects the toy, payment is made, but the child does not care whether payment is accomplished by cash, check, or credit card. Once the father decides to pay with a credit card, the details of money transfer and verification are handled by a combination of the card itself and the verification device. The services layer corresponds to the interaction between the child and the store; the payment policy layer corresponds to the father; and the payment mechanisms layer corresponds to the credit card used for the purchase.

We are using distributed object technology to implement InterPay. This technology is well suited for the high degree of modularity inherent in the InterPay architecture. It allows us to construct entities that directly correspond to components in the InterPay model. Payment agents, for instance, are implemented as objects. Our implementation generally accesses independent services, such as online information sources, through objects we call *library service proxies*. Each of these objects provides a message-based interface to the service it represents[1]. The methods of a library service proxy object ensure that the corresponding appropriate actions will be taken in the actual service. When we speak of "sending a message to a service," it should be understood that the message is being sent to a library service proxy which invokes the appropriate method to produce the proper behavior from the actual service.

The InterPay model is shown graphically in Figure 1. Ovals in the figure denote distributed objects. Entities shown in a cloud are outside services that are autonomous and whose interaction protocols cannot be changed. Links between nodes indicate communication paths which are implemented as messages sent between objects. When the communication involves an outside service, other links such as telephone lines, telnet connections or http transactions may be used.

At the payment policy layer, a *payment agent* negotiates for an acceptable mode of payment with the service's *collection agent*. Payment agents receive and process *invoices*, validate that the charge is to be paid, and invoke a particular *payment capability* that will complete the transaction. As the mediator between the user and the collection agent, the payment agent is a critical component that must be flexible, extensible, and reliable.

A collection agent is conceptually symmetric to the payment agent. A collection agent is associated with any party receiving money, and interacts directly with the payment agent. Its responsibility is to generate an invoice for a charge, then ensure that the payment for the invoice is of an appropriate form and is credited to the service's account. The collection agent is invoked whenever the service wants to levy a charge against a user. This might occur before a service request is started, before results are returned, or just because a certain amount of connect time has passed.

Each available means of payment, such as a Visa card, a First Virtual account, or DigiCash dollars, is encapsulated in a *payment capability*. Payment capabilities encapsulate both the data pertaining to the specific payment model, and the operations necessary to accomplish the payment transactions. Examples of payment model-specific data are Visa numbers or Dialog account numbers. An example of a model-specific operation is the response to an electronic mail confirmation request that is expected by the First Virtual payment scheme [9].

When a payment capability is invoked with an invoice, the payment agent has already established that the invoice is legitimate, so it is only necessary to pay the indicated amount. Each payment capability must handle the idiosyncrasies of payment using the method of its specialty. The payment capability must provide the collection agent with proof (e.g., a Visa confirmation number), allowing the collection agent to verify that its associated service has received its payment.

Just as payment agents use payment capabilities to accomplish the payment mechanism-specific activities, collection agents employ *collection capabilities* each specialized to one particular payment mechanism. Collection capabilities can examine proofs of payment passed from payment capabilities, and verify that the associated payments have actually been made. An InterPay system may contain many payment agents, collection agents, payment or collection capabilities.

Payment (and collection) agents need to implement financial strategies such as "do not bother me for transactions below 50 cents," "warn me if I spend more than $5 in a day or $50 in a month," or "I prefer to use DigiCash when possible." These strategies can be hard coded within each agent, but often they involve interactions with users at the time of payment. Thus, each agent may have an associated user interface to one or more users. For example, one interface we have implemented in our prototype (Section 5) is a "taxi meter" interface. This shows the accrued charges in much the same way a taxi meter does. In addition, this interface provides a set of "sliders" where the user can set limits for payments. The payment agent can also decide which user(s) it interacts with; e.g., the end-user receiving the information may not be the one making the payment decisions.

The InterPay architecture separates the task-related concerns from financial interactions (as shown in Figure 1). This simplifies the tasks of decision makers and programmers for both the information provider

[1]We have also created more general objects which provide a message-based interface to the World Wide Web. These are used to access services we have not constructed explicit library service proxies for.

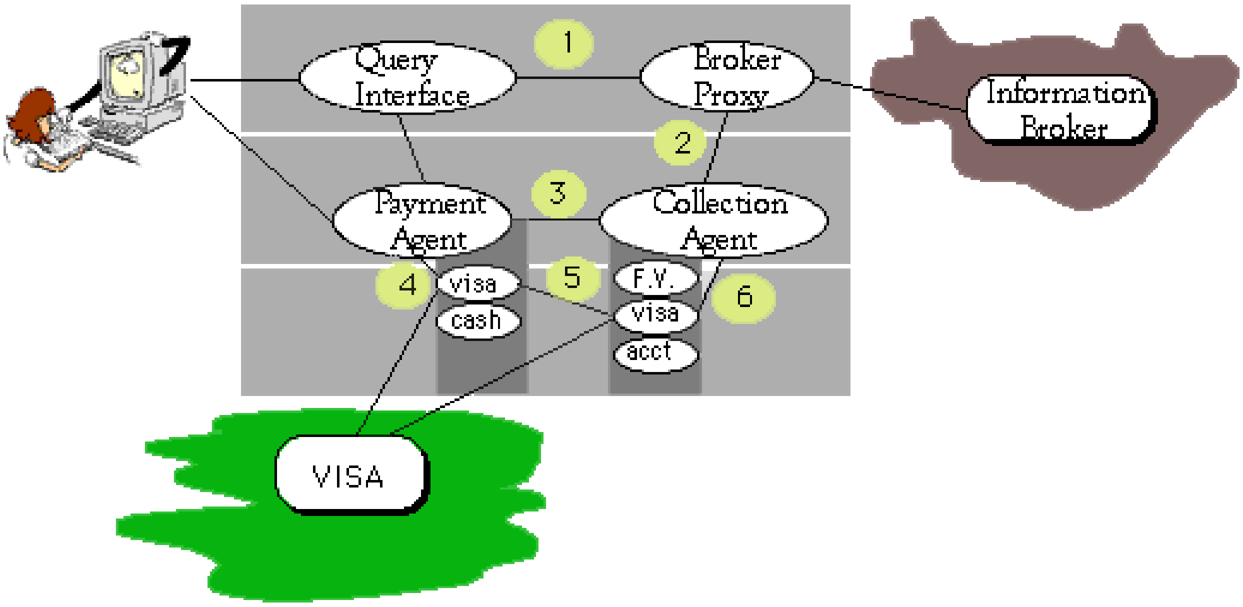

Figure 2: Interactions Among InterPay Components

and the consumer. This will be illustrated by the following detailed example, which shows the message flow and operations involved in a typical library search.

3 Extended Example

Joe, a computer consultant, is accessing a for-pay information broker that provides both search and retrieval services. From Joe's point of view, he interacts with two distinct components. One is his query interface, represented by the query interface object in Figure 2. It is an instance of a general class of user-level components which could be substituted for one another. For example, Joe might have available two separate query interfaces, one for very simple queries, and a second that supports a much more complex query model.

The second component Joe interacts with is his payment agent. For instance, Joe may have configured his payment agent (say by setting the sliders on his agent's interface) to warn him only if the cost of a transaction exceeds $10, or the total amount spent exceeds $500. Again, Joe could actually have more than one payment agent available. For example, Joe might have one payment agent for corporate clients, and another one for non-profit client organizations. The policies enforced by the two payment agents, or the user interfaces they present, would differ.

The InterPay architecture makes the query interfaces and payment agents interchangeable. That is, Joe could use the corporate payment agent with either interface, and the simple query interface could be used with either payment agent. Each component presents the same interface to other objects, regardless of its internal policies or the interfaces it presents to users.

3.1 Message Flow

We will now step through a particular search in detail. We will assume that Joe has configured his payment agent to use Visa where possible. He is making a request to the information broker, which charges $1.50 per requested citation. Figure 2 outlines the flow of messages between objects. There are six major phases to the communication, represented by the numbered arcs in Figure 2. These phases are:

1. **Set up the session and make a request.**
2. **Initiate a charge.**
3. **Send an invoice.**
4. **Validate invoice and agree on a payment mechanism.**
5. **Initiate the fund transfer.**
6. **Verify the payment and complete the transaction.**

We consider each of these phases in turn, describing the processing that occurs and the communication between the parties. The specific messages are described in the appendix, but the names are sufficiently evocative to suggest their purpose.

In the example we use the following abbreviations:

`QI`	the query interface
`BP`	the Information Broker's proxy
`PA`	payment agent
`CA`	collection agent
`PC`	payment capability (Visa in this example)
`CC`	collection capability (Visa)

The notation `QI: QI.SessionID = BP.StartSession(PA)` means that the query interface sends message `StartSession` to the Broker proxy object, passing a pointer to a payment agent as a parameter. The result is stored in the query interface's `SessionID` variable.

1. **Set up the session and make a request**. Joe makes a query, which gets directed to the query interface object. Joe's query interface contacts the Broker proxy to create a session which will allow the Broker to submit all future charges to the appropriate payment agent. Joe makes a query in this session, and warns his payment agent to expect a bill for the query.

   - `QI: QI.SessionID = BP.StartSession(PA)`
   - `QI: BP.AnswerQuery(QI.SessionID, "Digital Libraries and NSF")`
   - `QI: PA.ExpectCharge(QI.SessionID, "Digital Libraries and NSF")`

2. **Initiate a charge**. The Broker proxy relays the query to the "real" Information Broker, which returns say 16 relevant documents costing \$1.50 each. Before the Broker proxy releases results, it tells its collection agent to request payment for the citations from the payment agent. The Broker proxy generates and passes a `TransactionID`, which the collection agent will reference when the payment is complete. The `Charge` call is asynchronous, which allows the Broker proxy to do other work while the payment is being processed.

   - `BP: CA.Charge(PA, SessionID, TransactionID, $24.00, "Results for Digital Libraries and NSF Search")`

3. **Send an invoice**. The collection agent creates an invoice, indicates what means of payment it is willing to accept, and asks for payment from Joe's payment agent. The invoice contains the Session ID, amount, description, and a reference to the collection agent issuing the invoice.

   - `CA: PA.AuthorizePayment(Invoice, [FV, Visa, ACCT])`

4. **Validate the invoice and agree on payment mechanism**. Joe's payment agent needs to determine if the invoice is for a legitimate charge. In our example, this amounts to checking whether the query interface has filed an `ExpectCharge` matching this transaction, which it has. Alternatively, the payment agent could have required that the query interface specify a maximum amount that may be paid for the query; or it might have required only a valid session ID (for further discussion, see Section 5.4).

   Next, the payment agent has to decide if the amount of the charge is acceptable. Since nothing was specified in the `ExpectCharge` call and the requested \$24 exceeds the \$10 threshold that Joe established, the payment agent seeks verification before proceeding with payment. Joe responds affirmatively to the dialog box raised by his "taxi meter" interface, so the payment agent proceeds to select which financial instrument it will use to pay the invoice from the list supplied by the collection agent. In our example, Visa is acceptable for the collection agent and is the user-preferred method, so it is selected.

- `PA: if not PA.Payable(Invoice) raise exception`
- `PA: PC = PA.SelectCapability(Invoice, [Visa, FV, ACCT])`

5. **Initiate the fund transfer**. The amount of the charge and the payment destination are passed to the Visa payment capability, along with a pointer to the collection agent's Visa collection capability. The Visa payment capability performs the steps specific to a Visa transaction, such as obtaining the Broker's Visa account ID from the collection capability and contacting the Visa company via phone or other channels to request the transfer of funds. Visa returns a confirmation of the transaction, showing the amount that was transferred between accounts.

- `PA: PC.TransferFunds(Invoice)`
- `PC`[2]`: PC.HereIsYourProof(Invoice, Proof)`

6. **Verify the payment**. The payment capability passes the receipt from Visa on to the collection capability, which verifies that the money really was transferred to the collection agent's account. In this example, that verification process might include checking Visa's digital signature [2] on the receipt of the transfer. The collection capability informs the collection agent that it considers the invoice paid. The collection agent issues a receipt to the payment agent, and informs the Broker proxy that the payment has been made, without elaborating on payment details. The service consequently releases the search results. (Additional queries could follow the same procedure, possibly using the same session identifier.)

- `PC: CC.CheckThis(Invoice, Proof)`
- `CC: CA.ProofChecksOut(Invoice)`
- `CA: PA.HereIsYourReceipt(Receipt)`
- `CA: BP.PaymentAccepted(SessionID, TransactionID)`
- `BP: QI.HereAreYourResults(Results)`

This example shows that the details of payment have been effectively separated from the task layer. The service merely requested that an amount be charged and received notification when the payment process was completed. The service was oblivious to the details of the payment request, negotiation about payment mechanisms, or missteps along the way to eventual payment. The query interface was similarly uninvolved with the payment, delegating all details to the payment agent. The payment agent and collection agent were isolated from the particulars of the selected payment mechanism, ignorant of the means by which Visa was contacted, and the challenge/response verification which convinced the collection capability that payment was complete. Since the payment mechanisms are completely separate objects, it is possible to add new ones without changing the implementation of either payment or collection agent. The specification of the charging policy (how much to charge and when) still lies within the service, allowing a service provider ultimate control over its business model.

4 Other Scenarios and Variations

In this section, we introduce some variations on InterPay. We also present two more scenarios to demonstrate how the InterPay model generalizes to various situations common in the context of work involving the use of digital libraries. The first scenario shows how our framework can handle intermediaries, bulk-rate issues, and privacy issues. The second scenario explores how third-party payment might be handled, and gives one approach to addressing the social issue of information access for the disadvantaged.

[2] This message is sent by the part of the payment capability which handles the telecommunications with the Visa company.

4.1 Unification: Financial Agents

In our initial implementation, we had separate classes for payment agents and collection agents. As we implemented methods in the payment agent to handle refunds, we discovered that we were repeating capabilities that collection agents already had for receiving payments. Therefore, we are extending InterPay to include the notion of a *financial agent* that unifies the payment and collection agents. Payment agent and collection agent are simply roles that specific financial agents play in each transaction, i.e., a financial agent might serve as the payment agent for one transaction and a collection agent in another.

A financial agent should handle all of the messages of payment agents and collection agents. When acting as a payment agent, it would contain many payment capabilities, but few collection capabilities. These collection capabilities (refund capabilities) would correspond to refund mechanisms, such as the ability to receive change from a digital cash system, or to rescind a credit card charge. Notice that the refund capability for a credit card is different than a full collection capability. In particular, a refund capability does not give one the ability to charge others' credit cards.

4.2 Money Conversion

A type of library service that these general financial agents are particularly well-suited to handle is money-changing. A money-changing service could convert currencies (receive payment in dollars and pay out yen), or payment types (receive DigiCash and pay First Virtual). The wide availability of such services would reduce the need for each library service to support many different payment and collection capabilities.

4.3 Scenario: Library Intermediary

Library accesses frequently go through intermediaries that charge their clients and pay the ultimate sources. For example, Jane, who works for a large computer manufacturer, wants to use an information broker to find out more about a new algorithm she heard about in a seminar. Her department has an account for such searches. Jane could access the information broker directly, charging her department. However, her company's corporate library also provides access to the broker, and has a bulk rate discount. Thus, Jane prefers to go through the corporate library. (The library could also provide "value added" services, for instance, providing links to the library's holdings when appropriate.)

Figure 3 shows how this scenario could be implemented in InterPay. The payment agent for Jane's department is on the left; the one for the corporate library on the right. When the information broker wants to collect on an invoice for Jane's search, it contacts the library payment agent, which perhaps tenders a blanket purchase order number as payment. In turn, the library charges Jane's payment agent, perhaps a higher amount to cover its costs.

The structure of Figure 3 generalizes to multiple intermediaries, for example, when the information source itself collects information from other sources. This allows for site licenses and discounts to be implemented within InterPay. This example also illustrates how to provide privacy without anonymous payment, under the assumption that the library can be trusted not to forward the user's identity. This is a fairly reasonable assumption, as many libraries already recognize this issue, and allow users to protect the privacy of their information accesses.

4.4 Scenario: Student on Scholarship

Harris is an undergraduate student at a private university on scholarship. As part of his scholarship, he was given access to a payment agent to use for school-related information access. The granting agency controls the payment agent and has programmed it to deny expenses for extracurricular activities.

This payment agent is opaque to the student. No "taxi meter" will appear when Harris accesses the digital library. He cannot tell whether his purchases are being paid for on a purchase order or with digital cash. However, the agent may have an interface to the scholarship agency in case someone there needs to authorize expenses or set policies in a dynamic fashion.

The payment agent can implement a variety of policies for the scholarship agency. For example, if the campus bookstore has a "branch" for course-related materials only, the scholarship agency might take

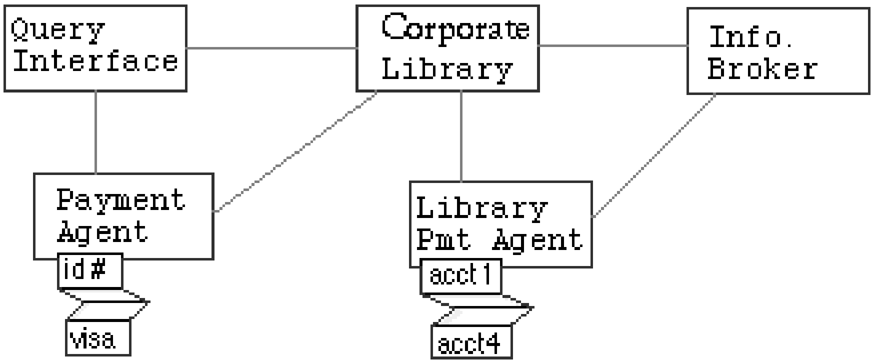

Figure 3: Corporate Library Pays for Client Searches. (Boxes in this figure represent a user's view of services, so collection agents are not shown.)

advantage of this by programming the payment agent to allow purchases only from that restricted branch. Another possibility is to require machine-parseable descriptions at the payment policy layer, such that agents in general can be programmed to distinguish the digital equivalents of course materials from newspapers or pornography. If such labelling were instituted, the scholarship agency could take advantage of it when configuring the payment agent.

5 Discussion

We have described the InterPay architecture and shown how it can be used in several scenarios. In this section, we step back to discuss the current implementation and some issues we are addressing, including security, performance, and the notion of trust.

5.1 Prototype Implementation

As part of Stanford's work for the NSF/ARPA/NASA sponsored initiative on Digital Libraries [4], a prototype of the InterPay architecture was implemented in a combination of C++ and Python, using Xerox PARC's ILU distributed object system (a CORBA implementation) [1]. A single program accesses three different existing information sources: Knight-Ridder's Dialog Information Service, the World-Wide Web (via WebCrawler), and INSPEC through an on-campus access point. The prototype uses different charging policies for each service. The payment capabilities are not currently hooked in to real financial institutions, but work only with place holders.

The interface to the system has two components, a World-Wide Web browser and a Python/Tk (X-windows) interface to the payment agent. The browser accesses a Python script which has an ILU module linked in, and which generates search forms and formats pages in response to searches. The payment agent interface lists transactions, shows the current amount spent, and provides an easy way to set the threshold levels, such as at what level to require confirmation ("don't bother me for less than \$.50") or the maximum to spend. We expect this interface to evolve significantly as the system becomes more complex.

5.2 Security

We find two security vulnerabilities in the system described, but believe that both of them can be taken care of with features added at the level of the distributed object system. First, there is the hazard that an eavesdropper can listen to messages that travel over the network and use the captured information to make fraudulent charges. Since any piece of data travelling over the network is potentially valuable and costly if disclosed, a general solution that goes beyond the limited scope of financial transactions is indicated. The providers of our distributed object system have recognized the value of encryption, and are working to offer strong encryption as an option for security. Key management for encryption schemes may be handled by trusted network services, such as the ones providing name services for locating services in a distributed environment.

A second kind of threat potentially comes from impostor objects pretending to hold financial authority. For instance, a service may attempt to impersonate the user and declare all invoices from that service to be payable without question. There is a range of similar attacks all relying on the ability to "spoof" a different origin. Therefore, we rely on the object system to provide authentication of message senders. With this underlying capability, method implementations can take into account the caller, providing different access levels for trusted users or agents. We are also investigating other encryption capabilities, such as SSL [6].

5.3 Performance

One potential disadvantage of a layered architecture is its performance overhead. In addition, the distributed object machinery (e.g., ILU) can add more overhead, as compared to compiled function calls. However, a preliminary performance study indicates that the costs may be tolerable.

We conducted an evaluation of raw TCP, ILU, and HTTPD on four UNIX machines ranging from a fast DEC Alpha to a 486-25 laptop (running Linux). Our results are presented in [5]; here we briefly comment on some of them. On unloaded machines we can achieve between 6.7 and 430 ILU method calls per second. This is about 8 times slower than raw TCP messages flowing over an open socket, and compares to the maximum of 6 WWW pages per second which we obtained on an RS6000 (the best result). We estimate that the InterPay protocols implemented on ILU add a 10% response time overhead to a simple transaction that accesses a single WWW page. This is assuming our financial operations add 8 ILU method calls, and does not consider delays incurred by the actual financial services.

There are several ways in which this overhead can be reduced. For example, as ILU matures, its performance will likely improve. Also, some of the financial operations can be done concurrently with the library accesses. A variety optimizations can be applied to payment and collection agents, some of which are mentioned below.

5.4 Trust

Once clients and services have an established history of interaction, they may opt to modify their transactions to reflect this increased trust. Requiring payment verification before any information is returned guarantees payment, but forces the client to wait longer to get results. With trusted parties, the service might perform the requested work right away, rather than waiting until payment has been approved, under the assumption that a user with a good history is unlikely to deny payment later. Similarly, a user might be willing to authorize payment for all invoices from a given service (not requiring an explicit ExpectCharge call for each one) on the grounds that the service is unlikely to risk the loss of all future business to submit one bogus invoice.

Another variation possible if trust is established between client and service is for payment agents to refrain from making payments themselves (via a payment capability). If a service is trusted, the payment agent can instead pass account information, such as a Visa number, to the service's collection agent. Payment could then be carried out by the collection agent, bypassing several interactions required by the standard protocol. This variation has the additional property that a service can feel somewhat more secure, because it does not need to rely on its ability to verify proof of payment. Our standard protocol does place payment control at the client side to avoid wide dissemination of confidential access codes, like credit card numbers.

6 Conclusion

We have described an architecture which allows many payment mechanisms to coexist, which supports many payment policies, and which simplifies user interactions in a for-pay digital library. These goals have been achieved by encapsulating the economic aspects of the library into a payment agent and a collection agent, so that the user and service are effectively isolated from the specific details of heterogeneous payment mechanisms. The specific details of payment mechanisms have also been encapsulated into objects, so that the payment and collection agents can be programmed by people who understand payment policies, but not necessarily payment details. Not only do these divisions simplify the interconnections among the modules, they also enable modules to be replaced, or new modules to be added. In this manner, the charging and collection functions are separated from the library-related services, and become first-class library services on their own.

The high degree of flexibility inherent in the InterPay architecture takes on added importance in the face of the ongoing debate about the place of fair use in digital libraries. Fair use laws limit copyright protection if copies of materials are used for the public good. The future of this law for digital libraries is under debate. Flexibility is required for InterPay to allow efficient future implementations of the resulting policies. We have implemented a prototype which currently accesses three online information services under varying payment policies.

Acknowledgements

We are grateful for the support of NSF, ARPA, NASA, and the partners of the Stanford Digital Library Project. In particular, Bill Janssen of PARC for ILU, James Kittock of Stanford for early discussions, and Vicky Reich and Rebecca Lasher of the Stanford Libraries for helpful discussions about fair use and library users. The prototype was developed on machines supplied by DEC, HP, and IBM.

References

[1] A. Courtney, W. Janssen, D. Severson, M. Spreitzer, and F. Wymore. Inter-language unification, release 1.5. Technical Report ISTL-CSA-94-01-01 (Xerox accession number P94-00058), Xerox PARC, May 1994. Available at ftp://ftp.parc.xerox.com/pub/ilu/ilu.html.

[2] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(5):644–654, November 1976.

[3] DigiCash. Digicash brouchure. Available at http://www.digicash.com/publish/digibro.html, 1994.

[4] Stanford Digital Libraries Group. The stanford digital libraries project. To appear in CACM special issue on digital libraries, 1995.

[5] Scott W. Hassan. Stanford digital library testbed performance analysis, 1995. Available at http://www-db.stanford.edu/ testbed/ilu/iluanalysis.

[6] Kipp E.B. Hickman. The SSL protocol. Technical report, Netscape Communications Corp., kipp@netscape.com, February 1995. Available at http://home.netscape.com/info/SSL.html.

[7] Knight-Ridder Information Services, Inc., 3460 Hillview Ave.; Palo Alto, Ca 94304. *DIALOG Database Catalog*, 1994.

[8] B. C. Neuman and G. Medvinsky. Requirements for network payment: The netcheque perspective. In *Proceedings of IEEE COMPCON'95*, March 1995.

[9] L.H. Stein, E.A. Stefferud, N.S. Borenstein, and M.T. Rose. The green commerce model. Technical report, First Virtual Holdings Incorporated, October 1994. Available at http://www.fv.com/tech/green-model.html.

Appendix: InterPay Messages

Message	**Invoked by**	**Description**
Payment Agent Messages		
AddCapability	client program/user	Empowers the PA with an additional payment mechanism. For example, the user may provide the PA with a VISA number.
SetLimit	client program/user	Establishes a value for the parameter that determines when the PA automatically accepts or rejects invoices.
SetMaximum	client program/user	Establishes the amount at which the PA will refuse to pay for further services.
ExpectCharge	client program/user	Asserts that a service has been requested. This message causes the PA to accept an associated invoice for further processing.
Payable	payment agent	Determines whether a newly received invoice is legitimate. May depend on the fact that it is "expected", came from a recognized source, or was accepted by the user when interactively asked for confirmation.
SelectCapability	payment agent	Chooses a payment capability to use for an invoice which has been deemed payable. May be a function of the amount, source, or user-declared preference.
AuthorizePayment	collection agent	Presents an invoice to the PA.
HereIsYourReceipt	collection agent	Presents final proof of purchase, to indicate that the payment authorization has succeeded.
Collection Agent Messages		
Charge	service	Specifies a PA, an amount, and a description of service rendered. The CA bills the PA for the amount, and notifies the service when the money has been received.
WillYouAccept	payment agent	Allows the PA to see if the CA can handle other payment types in addition to the preferred set specified in the invoice.
ProofChecksOut	collection capability	When the collection capability sends this message, the collection agent is able to inform the LSP of payment.
Cancel	payment agent	Indicates the PA does not intend to pay the submitted invoice. (Not essential, since not receiving payment within an allotted also denies payment.)
Payment Capability Messages		
TransferFunds	payment agent	Causes the payment capability to effect actual transfer of funds to the service provider.
HereIsYourProof	payment service	Presents proof of funds transfer to the payment capability.
Collection Capability Messages		
CheckThis	collection agent	Causes the collection capability to check whether an enclosed proof of payment is valid.