Payment Manager — Overview

J. Abad Peiro, N. Asokan, M. Waidner*

SEMPER Activity Paper 212ZR054
21 Mar 1996

ABSTRACT

The payment manager is that component of the transfer layer of SEMPER that is responsible for management of the user's purses. We provide an overview of the architecture of this component. The specificaiton of the payment manager, including the interfacess required and exported by the payment manager will be described in a separate document.

TABLE OF CONTENTS

1 INTRODUCTION 2
2 CONCEPTS 3
2.1 Payment Systems and Terminology 3
2.1.1 Players 3
2.1.2 Payment Models 4
2.1.3 Scope of a Payment System 5
2.1.4 Receipts 6
2.2 Electronic Purses 8
2.2.1 Concept 8
2.2.2 Attributes of a Purse 9
2.2.3 Registration of a Purse 12
2.2.4 Access Control 12
3 SERVICES 13
3.1 Session and Operation Management Services 14
3.2 Purse Management Services 15
3.3 Information Services 15
3.4 Value Transfer Services 15
3.4.1 Payment 15
3.4.2 Currency Exchange 16
3.4.3 Reversal payments 17
3.4.4 Cash-specific Value Transfer Operations 17
3.4.5 Authorisation and Capture 18
3.5 Recovery and Dispute Management Services 18
3.6 Other Services 18
3.6.1 Extended Information Services 18
3.6.2 Extended Purse Management Services 19
4 SECURITY 19
5 WHAT COMES NEXT? 20
6 REFERENCES 20

* IBM Zurich Research Laboratory, Rüschlikon, Switzerland — Phone +41 1 724 8220, Fax +41 1 710 3608, e-mail {jla,aso,wmi}@zurich.ibm.com>, www <http://www.zurich.ibm.com/>

1 Introduction

The payment manager is part of the transfer layer of *SEMPER*. This layer provides services for "transfers" of money, certified documents, and pure information, and for the management of the underlying systems used by *SEMPER* to provide these services. All transfer services are controlled by the transfer manager (see Figure 1). See [Waid 96] for an overview of the *SEMPER* architecture.

The transfer manager uses the payment manager to carry out transfers of money. Some services of the payment manager are provided directly to the users of the transfer layer (e.g., purse management services). Other services (e.g., payment functions) may be directed through the transfer manager.

Like in any generic service block in *SEMPER*, the payment manager in the payment service block will utilise "modules" (external payment systems) to implement the payment services. A number of digital payment systems have been recently proposed and/or implemented (see [JaWa 96] for an overview). The payment service block provides, to the extent possible, a generic payment service based on a unified model that covers, hopefully, all the different payment systems available. There is an internal interface for each supported *payment model.* Specific payment systems are mapped to the corresponding internal interface by means of a system-specific *adapter.*

As a matter of principle, the payment manager will always try to use the services provided by the actual payment system. Only if a required service (e.g., "receipts" or "reversal payments") is not available will the payment manager execute a generic protocol or raise an exception.

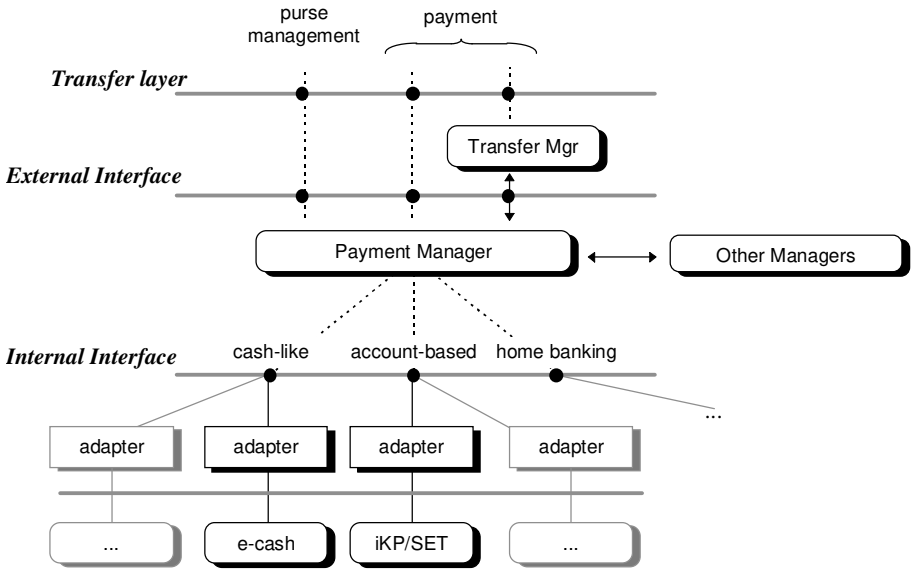

Figure 1 Interfaces of the Payment Manager

2 Concepts

2.1 Payment Systems and Terminology

2.1.1 Players

The basic set of players involved in each payment system consists of (see Figure 2)

Payer	The player who wants to send money to the payee.
Payee	The player who wants to accept money from the payer.
Issuer	The payer's bank which links the electronic payment to a real transfer of money. The issuer may give "electronic money" to the payer to pay with, or may send the money to the payee's acquirer.
Acquirer	The payee's bank which finally materialises the electronic payment.
Arbiter	Settles a dispute between participants in the system.
Attacker	A player who attempts to violate the security and integrity of the system; the attacker can be an **inside attacker**, who is a legitimate participant in the system (participating in one of the roles already listed above) or an **outside attacker**.

Instead of "issuer" and "acquirer" we will also use the term **bank**. In several payment protocols, the "payer" and the "payee" are referred to by more specific terms like "customer" or "buyer" and "merchant" or "seller", respectively. We prefer the terms "payer" and "payee" since on the payment system level there is no reason to limit the application to just purchases.

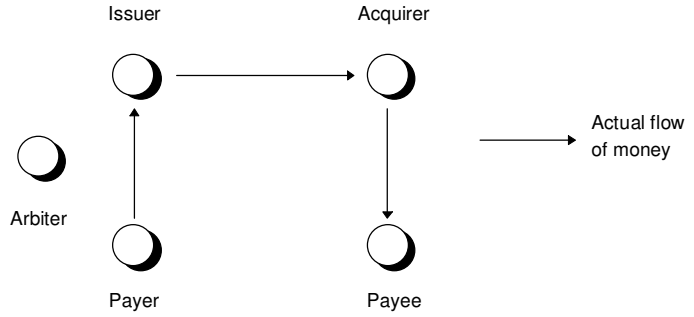

Figure 2 Players of a Payment System

Certain payment systems might involve more players, e.g.,

- registration and certification authorities for all other players, or
- trusted third parties that enforce receipts for payments.

In most systems, the presence of the "arbiter" is not explicit, i.e., even if the necessary proofs of evidence are produced dispute handling is done outside (and often not specified).

2.1.2 Payment Models

We classify payment systems according to the necessary flows of information between the players. Figure 3 lists, without claiming completeness, the four most common payment models and their information flows (see also [JaWa 96]).[1]

One criteria for distinction is whether there is a **direct** communication between the payer and the payee or whether the communication is only **indirect.** In the latter case, the payment operation is initiated by one party and involves only the banks. The other party is notified by its bank at the completion of the transaction. An example of the former is paying by cash or cheque. An example of the latter is paying by means of a standing order or wire transfer.

All proposed Internet payment systems implement direct payments. Consequently, *SEMPER* will support only these systems initially.

According to the temporal relationship between the time the payment initiator considers the payment as finished, and the time the money is actually debited from the payer's account, one can distinguish between

- pre-paid (or "cash-like") payment systems, see Figure 3a,
- pay-now payment systems, and
- post-paid payment systems, see Figure 3b.

The latter two are quite similar[2]. We will refer to them as the **account-based**[3] model. Therefore, in the rest of this document, we will use the terms **cash-like** and **account-based** to refer to the two models.

In an **on-line payment system,** the payer and the payee are connected on-line with a third party (typically issuer or acquirer) while in an **off-line payment system** the payment is a transaction between two parties only. On-line payment systems are often considered more secure since the third party can ensure that the payer actually possesses the money they want to transfer to the payee. In most off-line electronic payment systems, double spending of money is prevented by tamper-resistant hardware (smartcards or "electronic wallets"). As a second line of defence, some systems ensure that double spending would be provable, subject to the usual cryptographic assumptions, in case it happens (even for systems that offer anonymity otherwise), e.g., [BBCM 94].

1 One should not cofuse the flow of information with the flow of money. While there are many possibilities for the former, the latter is always as shown in Figure 2.

2 There *are* differences between the two models. Our current opinion is that these differences are minor. We can of course treat the two models separately should it become necessary.

3 Certain prodcuts of the cash-like model (e.g., ecash) may also happen to have corresponding accounts. However, this is an implementation "add-on." A barebones cash-like system need not have a corresponding account in a bank.

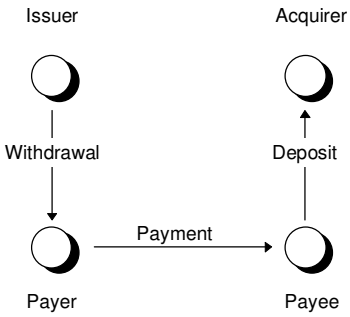

Figure 3a Direct cash-like system

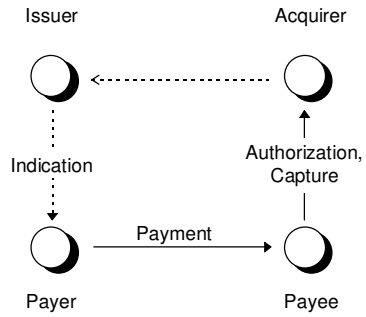

Figure 3b Direct account-based system

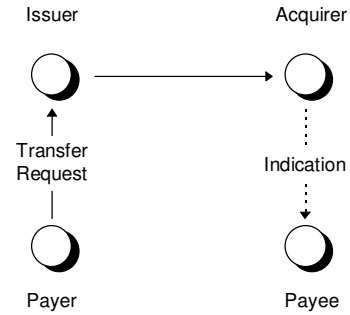

Figure 3c Indirect push system

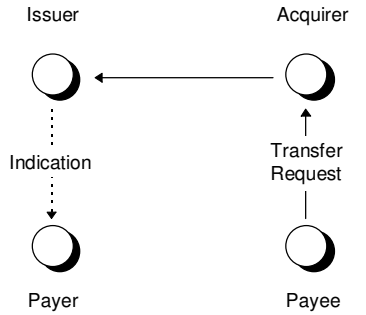

Figure 3d Indirect pull system

Figure 3 Payment Models (dotted lines indicate flows that may be external to the digital system)

2.1.3 Scope of a Payment System

A payment system deals with all operations that have to do with controlling the movement of money from the payer to the payee *inside the digital system*. Besides the obvious operation "transfer money from *A* to *B*," this includes operations for monitoring what has happened so far, for cancelling operations, exchanging currency, etc.

A payment does not imply the purchase of any goods. It is just a transfer of money from the payer to the payee. The reason why a payment is performed must be considered outside the payment, but a description of that reason might be linked to the payment data (as it is done in *iKP*, for instance).

The **payment manager** is that object on the transfer layer of *SEMPER* that manages all means of payments that are available for a user. The payment manager supports different payment models as described in Section 2.1.2, in a uniform way: The differences between payment systems implementing the *same* payment model should be hidden. The interfaces to different payment models should be as uniform as possible.

The functionality of the payment manager is described in a separate document; here we provide an overview only. Before proceeding, we explain some terminology:

Purse The primary data object in the payment manager is a purse. A purse is an object that contains all the user's information on a specific instance of a specific payment system (we could also call this an "account object"); for more details see Section 2.2.

Operation Primitive services offered by the payment manager are called operations. Some operations are atomic and can not be interrupted. Interruption of these atomic operations will either not happen or either will roll back the system to the previous state.

Session A sequence of related operations can be grouped together into a session. The distinguishing feature of a session is the context shared by its constituent operations.

We distinguish five classes of services:

1) *Purse management services* allow the user to set-up, configure, manage, and delete purses. Purse management includes access control to the purse. Some purse operations might require authorisation for each critical operation (e.g., to insert a smartcard and enter a passphrase for each withdrawal) while others may not have any access control.

2) *Session and operation management services* permit related operations to be bundled together by setting up a common context. Each payment manager operation can take a reference to an already established context as a parameter.

3) *Information services* permit the retrieval of information on the state of the payment manager, or a specific purse: e.g., a list of previous operations, the last statement of account received, or statistics on all payments received and made in a certain period of time.

4) *Value Transfer services* constitute the core of any payment system:

   - Moving electronic value from a payer's purse to a payee's purse. The payer might request a receipt from the payee or from the acquirer, depending on the selected payment model and level of security.
   - Moving electronic value back from payee to payer, i.e., in reversal payments.
   - Converting real money into electronic value ("loading") or vice versa ("deposit"); this is relevant only in the cash-like model.
   - Moving electronic value from one of the user's purse to another, e.g., currency exchange, (maybe also converting from one payment system to another; see Section 3.6). Note that currency exchange is not a unilateral process but one that requires running a protocol with the bank.

5) *Recovery and dispute management services* support the user in case something has gone wrong. This includes services to recover from a crash of the user's system or an interrupted atomic operation, but also to retrieve or verify proofs of evidence (like signed receipts or statements of to account) that must be shown to a third party.

2.1.4 Receipts

Receipts prove that a payment has actually happened. Technically, they are non-repudiation tokens, i.e., pieces of information that would convince a third party. Typically, receipts are implemented by digital signatures.

By itself, a "receipt" determines neither who issues that — the payee, the acquirer, or the issuer — nor what it means — payment data received, payment authorised, or money actually transferred. Additional information (e.g., a contract agreement, a refer-

ence to the protocol that resulted in the receipt, etc.) must be provided along with the receipt in order to give it meaning. The exact nature of the additional information required depends on the situation.

Also note that many payment systems do not support strong non-repudiation but only some sort of "secure acknowledgement."

Therefore we consider several classes of "proofs." In general, an **acknowledgement** does not give non-repudiation while a **receipt** does.

- **Local acknowledgements** are generated by the payment manager locally in case the action has been successfully completed locally, e.g., as non-error return codes. They do not prove anything to any other party.
- **Remote acknowledgements** are generated by the payee in case the payment has been successfully completed. It still does not prove anything to a third party, not even to the payer (since the origin is not authenticated). Only if an expected remote acknowledgement does *not* arrive, does the dissatisfied payer know that something went wrong.
- **Authenticated remote acknowledgements** are remote acknowledgements securely authenticated by the payee but without giving non-repudiation.
- **Signed receipts** are generated and digitally signed by the payee, giving non-repudiation of receipt of a payment.
- **Notarised receipts** are remote acknowledgements that give non-repudiation of receipt of payment based on a trusted third party (e.g., the acquirer). In addition, they might be signed by the payee, i.e., there are also **notarised signed receipts.**
- **Enforced receipts** are receipts that give non-repudiation of submission, i.e., the payer can prove that they have submitted the payment but it is not clear whether the payee has actually accepted it.

The same classification can also be applied to other "receipts," e.g., for receipts of deposits.

Obviously, the only non-trivial scenarios are the notarised and enforced receipts. Notarised receipts can be created as suggested, e.g., in the proposed (but not very stable yet) ISO standards on non-repudiation services.

Figure 4 shows an example design of a payment with *enforced receipt*. The protocol is optimistic, i.e., it involves a third party only for exception handling. *B* is a payer, *A* is a payee and *TP* is a third party trusted by both. *B* could act anonymously. If both *A* and *B* are honest then *TP* is not involved in the protocol (i.e., this is an optimistic protocol for fair exchange).

Initially *B* pays an amount to *A* [1] requesting a receipt for the payment. If *B* does not get a receipt within a certain time [2] then *B* decides to send a résumé of the payment (the "payment transcript") to *TP*. The *TP* verifies the validity of the payment and forwards it to *A*. Note that there is a limit to what the *TP* can verify. For example, in the case of payments by electronic coins, the *TP* should verify that the coins are plausible and record the payment transcript (perhaps after signing it). The *TP* will not verify the validity of the coins. If the coins turned out to be invalid, an arbiter outside the system can make use of the certified payment transcript to resolve the dispute.

If *A* answers with the requested receipt, *TP* forwards it to *B* — and the problem is solved [3]. If the payment was valid but *A* does not answer in a limited time (the second clock in Figure 4) *TP* generates a receipt and sends it to *B*.

Several payment systems, e.g., *iKP* can already provide the receipt in Step [2], but usually do not enforce it.

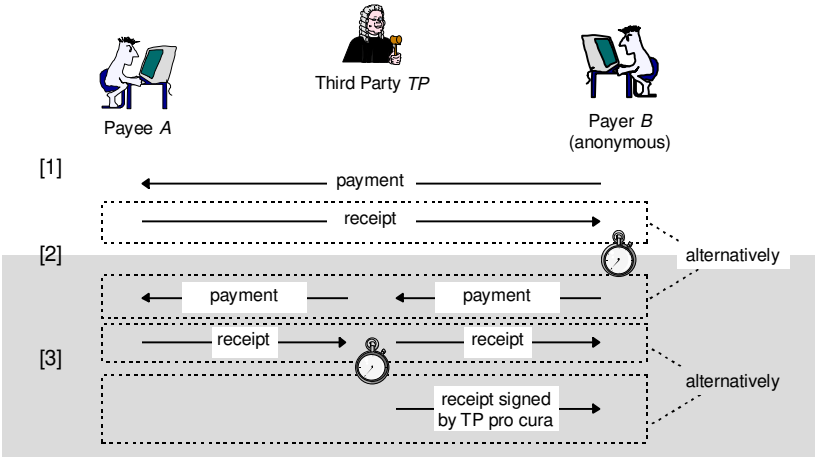

Figure 4 Payment with Enforced Receipt

2.2 Electronic Purses

2.2.1 Concept

Purses are those objects that represent the actual instances of payment systems available to the user. The payment manager controls the purses and operations on purses.

A purse does not correspond to a specific payment protocol (like *iKP*) or system (like *Eurocard/Mastercard*) but to an actual means of payment (e.g., to a *Eurocard/Mastercard* issued by *BGL* and supporting on-line payments based on *iKP*). If a physical payment system already has the notion of “purses” or “accounts,” we will map each of these purses or accounts to a separate purse in the payment manager. We allow purses to be grouped together when they share some attributes.

In the cash-like payment model, purses can be considered to “contain” electronic money. In the account-based model, the purses can be considered to be an electronic notebook for issued cheques or credit payments.

From an architectural point of view, a purse sits between the payment manager (that invokes the methods of the purse) and the payment systems that have been installed (the methods of which invoked by the purse).

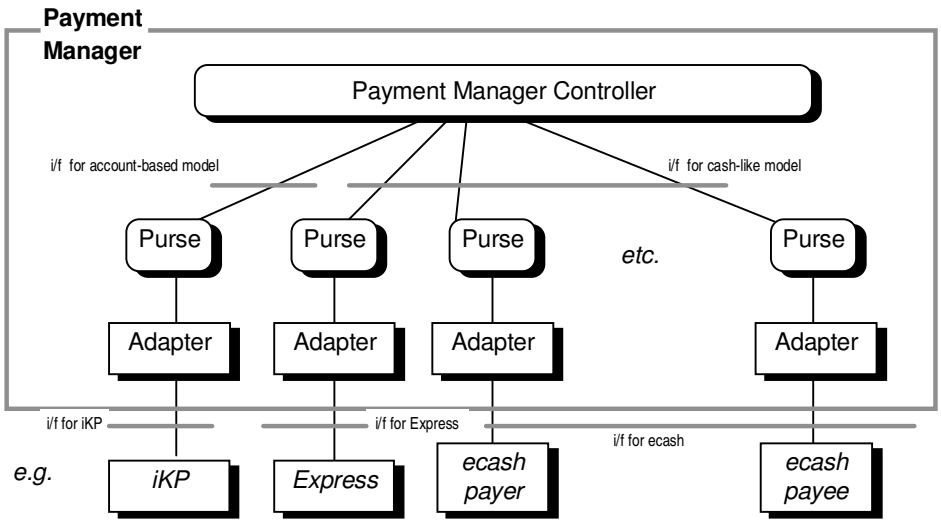

Figure 5 Location of Purses in the Architecture

An **adapter** maps the model-specific internal interface of the payment manager to the actual interface exported by the payment system.

In an object-oriented implementation, the internal interface will correspond to a generic model-specific class. A system-specific adapter will be a subclass of this which instantiates and extends this generic class. A purse will be an object of the latter subclass. As mentioned before, there will be a purse for each distinct means of payment available to the user.

In a procedural implementation, a purse is an instance of model-specific data structure. Again, there will be a purse for each distinct means of payment. The adapter will consist of a system-specific code module (that maps the functions of the internal interface to the functions of the system interface) and purse-specific data.

A further classification of purses can be made based on the subset of operations supported by the purse[4]:

- **pay-only purse** for a purse that can be used to make payments only, but that cannot receive money from a payer in a payment;
- **accept-only purse** for a purse that can be made to receive payments only;
- **pay-and-accept purse** or just **purse** for a purse that can be used for making and accepting payments.

Note that most current payment systems do not support pay-and-accept purses. Money received by the payee in an accept-only purse must be deposited first before the money can eventually be used for a new payment.

2.2.2 Attributes of a Purse

Each purse is characterised by a list of attributes. The type, role, state, and behaviour or purses are described and determined by the values of its attributes. There might be purses with very specific attributes (e.g., interest rates of accounts). Here we consider only the generic attributes that are present for each purse. The values of some attributes are fixed for the whole "life" of the purse (e.g., brand, bank, name) while the val-

4 Sometimes different names have been used elsewhere in the literature: e.g. an accept-only purse is sometimes referred to as a *till*. We feel that the names used here are less ambiguous.

ues of other attributes can vary. The payment manager will enforce concurrency control on operations that attempt to change the values of attributes.

Certain attributes may be shared by a group of purses. Such an attribute can optionally point to an "group attribute" instead of having a "local" value, indicating that the value of the group attribute should be used instead. A **group attribute** is similar to a purse attribute (purse attributes are listed below -- note that some purse attributes are inherently unshareable). In addition, it has an attribute "**group members"** whose value is a list which contains distinguished names of all the purses that have made a reference to this group attribute. Each purse keeps track of which of its attributes are shared and the identity of the group attribute for each of the shared attributes.

The first set of attributes identify the purse and the payment system, in an economic and technical sense:

Name Distinguished name of the purse, used by the payment manager.

Bank Issuer/acquirer used for this purse. "Bank" might include several attributes itself, e.g., a communication address, a certificate.

Payment system and brand name, e.g., "Eurocard". Must agree in both payer and payee.

Expiry Time of expiry for the purse. The purse creation and purse update operations will set the value of this attribute.

Status The purse status describes, as the name indicates, the status of the purse, i.e. if the purse is active or not, if it is saved or not, if it is waiting or not, etc. During the implementation phase we may decide to collect this attribute for all purses somewhere in the PM, although here we are describing attributes of the purse, so it looks natural to describe this attribute here.

The following attributes describe the *type* of payment system:

Model Payment model supported, currently "cash-like" or "account-based".

Operations Operations supported by this purse. The set of operations is different depending on model and role. Note that the information whether this is a pay-only / accept-only / pay-and-accept purse is implicit in the list of operations.

The following attributes describe limitations of the purse:

Currency Supported currency of the purse. Each purse supports a single currency. Currency conversion functions work between purses.

Amounts The total amount hold by the purse currently. There are several **counters** associated with payments with this purse. Each counter provides the following information:

Amount

In a cash-like system, this is the "stored value." In an account-based system, this can be the account balance or the available credit if the brand supports it. This has the sub-attributes: **current, maximum,** and **mini-**

mum. Notice that in account-based systems, the exact amount is maintained in the bank[5]. Therefore, the value of this attribute is only an approximation and needs to be synchronised with the bank periodically[6].[7]

Unlocked amount

This is the amount that can be directly used for payments without specific access control. Payments of larger amounts must be preceded by a specific "unlock" operation. This has the sub-attributes: **current, maximum,** and **minimum.**

Access control Each purse has an access control structure in the form of an array whose elements correspond to purse operations. Each element in the array is a list of tuples consisting of the fields "Condition on operation", "capability ID", and "info/action." See Section 2.2.4 for more details.

Limits Each of the following attributes can have a list of tuples as value.

Available amount per operation:

In case of operations that move something into a purse or out of it, certain limits are defined (after a currency is fixed). This is a list of tuples where each tuple consists of the following: **operation, current, maximum** and **minimum.**

Available number of operations:

This attributes defines the number of times a certain operation can be performed at the current time. This is a list of tuples where each tuples consists of the following: **operation, current, maximum, and minimum.**

Certain purse operations will update some of these limits. For example, a payment may update the current amount, unlocked amount, current available amount per operation, and the current available number of operations. For now, we do not discuss these limits further except for noting that they depend on policy restrictions as well as inherent limitations and that they represent both restrictive limits and "guarantees." We hope to refine these concepts further in forthcoming revisions of this document.

The payment manager maintains informational data corresponding to each purse:

Operation records: A operation record contains all the relevant information about every operation that results in a change in the state of a purse; e.g: every operation that results in a change of *amount*.

---

5 The bank could make unilateral additions(accruing interest) or deductions (service charges) to the amount maintained in the account. Further, the owner of an account could use it outside of the SEMPER infrastructure.

6 Synchronisation may not be possible in all cases. In such cases, the purse can either maintain an approximate amount or decide not to maintain the amount counter at all.

7 The available credit may not always be a unique value. It can depend on the parameters for the payment operation: e.g., the bank may decide to extend different credit limits for different classes of *payees*!

Statements of accounts: A statement of accounts for a purse can be generated from its operation records. In the case of purses that correspond to an account maintained in a bank, the bank may periodically send out statements of account. The payment manager is responsible for the handling of these statements (i.e., store them in the archive, expire corresponding operation records, etc.)

2.2.3 Registration of a Purse

Each purse must be registered with the payment manager before it can be used. Basically, this means that the purse object is generated and initialised. This will be a purse-specific process. Depending on the payment system, this process may also involve some form of registration with the bank. When the expiry date is reached, it may be possible to renew the registration.

In case the user can configure some parameters of the purse, or the underlying payment system can be used also without the purse, there must be functions to reconfigure the purse, or to reconcile the purse information with the actual state of the payment system.

2.2.4 Access Control

Access control is a critical functionality of the payment manager. We note the following in order to motivate our design:

1) Access control is required in the following cases:
   - Access to secret information required to use the underlying payment system (e.g., PINs, passphrases, credit card numbers, etc.). There may be several different pieces of such information.
   - Access to purse operations.
2) Even for the same purse operation, it may be necessary to control access differently depending on the parameters. For example, low value payments and high value payments may be controlled by different passphrases. The underlying payment instrument may or may not support such granularity.

We propose an access control structure as in Figure 6. Each operation has a list of tuples defining access control. The fields of a tuple are:

Disabled A single bit flag indicating whether the operation is currently disabled. This allows temporary disabling of certain operations without changing the more permanent access-control information associated with it.

Condition A logical expression in terms of the parameters to the operation (**true** is a trivial expression that matches all cases). When an operation is requested, the payment manager will sequentially walk through the tuples until it finds the first tuple whose condition is true for the current set of parameters. Example: “amount > $50”

Capability ID An identifier for the capability that is required to access the operation. This can be “none” for operations that require no access control. The capability may be presented by the invoker during invoca-

tion. Or it may indicate that the user (via the user authorisation manager) needs to be prompted for the capability. The ID also identifies the type of capability and therefore how it is to be used (e.g., decrypt the Info/Action field, verify against the Info/Action field etc.)

Info/Action The meaning of this field depends on the type of capability. It can be

- Information for verification of the capability (e.g., a hash of a passphrase), or
- Secret information (e.g., obtained by decrypting with the capability information) to be passed down to the payment system in order to carry out the operation, or
- A flag indicating that the corresponding capability itself can be passed down to the payment system in order to carry out the operation (i.e., there is no separate secret information), or
- empty, in case the operation does not involve any secret information. In this case access is granted if the corresponding capability has been submitted.

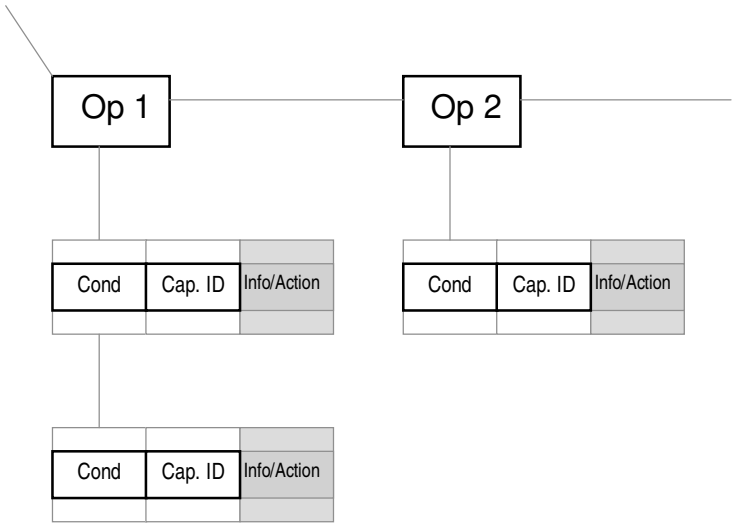

Figure 6 Access Control Structure

For example, if the payment system insists on receiving a password but the user doesn't mind storing it in the payment manager, the Info/Action field will contain the password and the Capability ID field will be "none."

3 Services

In this section, we describe the services provided by the payment manager. The following is structured like Section 2.1.3:

- Session and operation management services,
- Purse management services,
- Information services,
- Value transfer services,
    - Moving electronic value from a payer's purse to a payee's purse.

  - Moving electronic value back, i.e. in reversal payments.
  - Moving electronic value from one user purse to another.
  - Converting real money into electronic value ("loading") or vice versa ("deposit").
- Recovery and dispute management services.

We also mention certain "other services." It is not clear if they should be provided by the payment manager.

Linking names to payment identities: When calling the payment manager, payer and payee might have already established a business relationship using certain names (and related certified keys). These names are not necessarily the same as those used in the payment transaction (which are likely to be something like account numbers). In some cases (e.g., for receipts) the identities outside and inside the payment transaction must be linked.

There are two cases here. The invoker of the payment manager operation (e.g. a business application via the transfer manager) may want the payment manager to obtain receipts which have different identities than those used in the operation. In this case, the payment manager simply passes this request to the peer entity. It is up to the other side to decide whether the requested identities can be included in the receipt.

Alternately, the invoker may have specified a unique identity of the peer with which the operation is to be carried out. But the peer may instruct the payment manager to direct further interactions to a different identity (e.g. following a "pay to Company ABC" operation, the payment manager engages in negotiation with ABC in order to select a payment method; the result of this negotiation is "pay to account 343235 at Bank XYZ"). In this case, the payment manager should, perhaps optionally, be able to obtain and record non-repudiable proof of this "re-direction."

Non-repudiation of receipt of a payment: The payment manager allows the payer to request any kind of receipt but it does not enforce them.

This means that the payment system will be asked to provide a receipt on a best-effort basis. If the receipt for an opreation arrives the payment manager will forward it (or more exactly, a link to its location in the archive) to the invoker of that operation. If no receipt arrives in time the payment manager sends an error code to the invoker together with a link to the transcript of the payment operation. As described in Section 2.1.4 this can be used to enforce a receipt giving non-repudiation of submission.

We decided to implement the enforcement of receipts outside the payment manager since the mechanisms used are not specific for payments. In fact most the proposed mechanisms for non-repudiation of submission or delivery can be applied to any kind of transfer.

3.1 Session and Operation Management Services

A *session* is a sequence of primitive *operations* such as withdrawal, deposit, payment, information request that are linked together by a common context (i.e. certain parameters of these operations have the same value. Session management services consist of functions to create a session context, insert attributes into the context and modify their values, and to destroy contexts. Each operation can optionally refer to an already established context as a parameter.

Operation management includes services to cancel an active operation, report the current state of an active operation, etc. These services will be applicable only for a subset of all payment manager operations. If an atomic operation is interrupted (e.g., due to communication failure or system crash), it must be possible to reach a consistent state once again.

3.2 Purse Management Services

Purse management includes functions

- to create, initialise, and renew a purse (see Section 2.2.3),
- to modify the attributes described in 2.2.2,
- to synchronise certain purse attributes which are maintained elsewhere (e.g. in the bank or in the payment system), and
- to delete a purse.

3.3 Information Services

Information management includes functions to monitor sessions, operations and the state of purses.

This includes a general dump function that exports the complete transcript, or all proofs of evidence (non-repudiation tokens) linked to a specific operation so that they can be used in an external dispute.

Monitoring a session means the extraction of values in the session context. Monitoring an operation means getting information on the parameters chosen so far and on the state of the currently active operation.

Monitoring a purse means getting information about the attributes of the purse, about operations involving that purse, and about statements of account received for that purse.

An external service can use the raw data provided by the payment manager (such as operation records and statements of accounts) to compute and present a wide variety of statistical information. See Section 3.6 for further details.

3.4 Value Transfer Services

Value transfer deals with all kinds of movement of electronic value from one purse to another: payment, currency exchange, reversal payments, and for cash-like systems also withdrawal and deposit. Note that in addition to the *amount* attribute, value transfer operations may update other related attributes as well.

3.4.1 Payment

Payments are value transfers from a payer's purse to a payee's purse. The purses used must be compatible in the obvious sense, e.g.,

- the payer's purse must not be an 'accept-only' purse, the payee's purse must not be of type 'pay-only;'
- both purses must belong to the same payment system (indicated by the 'payment system attributes);

- the payment amount must be within the current limits defined by the purse attributes;

etc.

Note that the details of a payment operation and parameters might be different in the different models of payment (e.g., delayed authorisation in credit-card payment systems, or loading cash during a payment in a cash-like system). This will be discussed in detail in separate activity papers on the two models supported initially.

In case of micro-payments the payment operation itself consists of a sequence of a "start micropayment," several "do micropayment", and a "stop micropayment" operations. Some payment systems may already offer a time-based micropayment operation. A separate primitive "micropayment" will be provided by the payment manager to cover such a service.

Performing a payment might be subject to some sort of access control (usually, cash-like schemes do not restrict access to the payment; all other payment models do). This is handled as described in Section 2.2.4.

The details of the payment must be agreed on by payer and payee before a payment operation can be started. For this *negotiation* payer and payee enter a list of acceptable means of payment. The payment manager ensures that payer and payee agree on a common method, provided the lists of payer and payee have a non-empty intersection.

3.4.2 Currency Exchange

For account-based schemes there is no problem since payers can specify the amount in any supported currency they like.

For cash-like schemes, currency exchange has (at least) three meanings:

1. **Explicit currency exchange at the payer:** A payer moves an amount from one purse into the same or another purse, thereby changing the currency of the amount. Usually this makes sense only in pre-paid payment models. Note that this exchange cannot be done by the user acting alone but requires a run of a protocol involving the bank.
2. **Implicit currency exchange during payment:** Payer and payee do not agree on a common currency but on an exchange rate between the payer's and the payee's currency. The payee might need a guarantee that the rate will be valid for a certain time.
3. **Multi-currency amounts:** In a cash-like system one might consider 'coins' with denominations in more than one currency (e.g., a coin that is simultaneously 1 ECU and 1.7 DEM and 1.5 SFR, etc.).

All three possibilities were investigated by Project CAFE, with the result that there is no need to support (3) since the risk of changing exchange rates is considered too high in general. Possibility (2) need not be considered by payment system itself since it is more a question of how the amount to be paid is specified (but there must be a mechanism for guaranteeing a certain exchange rate for some time). This leaves us with possibility (1) which is in fact what is supported by most multi-currency purses, e.g., Express. In addition to (1), CAFE supports a "version" of (2): during payment, the payee can prescribe the amount requested in a foreign currency and an exchange

rate between the foreign currency and the currency of the CAFE wallet. The wallet would then compute the amount to be paid in its currency.

3.4.3 Reversal payments

In some cases, the payer might want to cancel a payment and get back the spent amount. If a payment system does not support to be reversed it can always be simulated by a payment in the reverse direction. But there are some limitations:

- The payer might have pay-only purses only.
- Untraceable payment systems usually protect only the payer's identity but not the payee's identity. A reversal payment may lead to the loss of privacy of the payer.

The first limitation can be relevant in account-based schemes (e.g., *SET, iKP* differentiate between payers and payees).

In untraceable cash-like schemes (e.g., *ecash*) one can solve this problems by making a withdrawal via the payee who wants to reverse a previous payment made to him by the payer. However, this may render certain fault-tolerance schemes ineffective (for example, a record of the withdrawal of ecash coins is preserved by the bank along with a link to the customer's real identity; this record will be used if the customer comes to the bank to recover his money after some catastrophic failure (e.g., loss of ecash coins).

SEMPER provides a generic reversal payment mechanism by trying to make a reverse payment (not necessarily using the same payment system used for the payment to be cancelled), but the payment manager will always try to use a system-specific reversal payment operation.

3.4.4 Cash-specific Value Transfer Operations

3.4.4.1 Withdrawal

The purpose of withdrawals is to enable payers to perform payments afterwards: A withdrawal (or load) moves "electronic cash" from an issuer into a local purse of the payer. The loaded amount might be debited to an account of the payer (or any other party in case of doing withdrawals via some other party, e.g., the seller in a reversal payment as sketched in Section 3.4.3), or it might be covered by real cash.

Some payment systems allow a transparent reload (subject to configured limitations) during the payment in case a payment cannot be performed.

Note that while access to the payment operation is not necessarily restricted, all known cash-like schemes require a PIN for withdrawals.

3.4.4.2 Deposit

The purpose of a deposit is:

- to authorise the payment, i.e., to verify that the payment is valid,
- to actually credit the amount to an account of the payee.

In on-line schemes both actions are part of the payment operation (but the payee might take the risk to postpone authorisation in case the acquirer is not reachable, or the

authorisation of low values is too expensive). In off-line schemes, authorisation is not required (i.e., it is done by the payer's purse which should be tamper-proof).

As mentioned in Section 3.4.2, the payee might request that the amount is changed into the payee's preferred currency for a fixed exchange rate. Most cash-like schemes do not allow the transfer of received electronic cash without intermediate deposit, i.e., deposit is almost always necessary.

3.4.5 Authorisation and Capture

These functions are specific to the account-based model and corresponds to "deposit" in cash-like schemes.

During "authorisation" the acquirer ensures that the payee would get up to the authorised amount in case the payee performs a "capture." During "capture" the amount is actually credited to the payee's account.

Authorisation and capture might be postponed (at the payee's risk), and both might be combined, so that they are done simultaneously.

Note that in credit-card like schemes the payment system is responsible to check that the amount authorised by the payer and those amounts for which the payee requests authorisation and capture are related in some reasonable way. The payment systems themselves do not prevent, e.g., that the captured amount is higher than the authorised one, and both are higher than what the payer actually wanted to pay.

3.5 Recovery and Dispute Management Services

Some of the functions are already mentioned somewhere else: recovering a single operation, procuring a receipt, etc. Additional services to recover the whole payment manager (e.g., after a crash) and individual purses are required.

Disputes of individual operations are part of these operations (e.g., there is not much need for a 'dispute payment' operation outside a payment).

3.6 Other Services

In this section, we describe some services that are related to the payment manager. It is not clear whether they should be in fact part of the payment manager. For now, we leave them out of the payment manager in the interests of simplicity. However, we outline the arguments for and against their inclusion in the payment manager.

3.6.1 Extended Information Services

By way of information, the payment manager provides a operation record (given the operation identifier) and statements of accounts. This information can be processed in various ways. For example, it might interest a user to know the average amount she has paid during a specified time period or the total amount withdrawn during a similar period.

One can argue that this service should be provided externally (by a commerce layer application, for example) for the following reasons:

- It is good practice to keep the design simple,

- There are existing tools to provide such services, and
- This service *can* be provided by an external application based on the information services already provided by the payment manager.

On the other hand, one can argue that it must be provided by the payment manager for the following reason:

- It is so easy to maintain certain pieces of statistical information that are most likely to be of interest to the user.

3.6.2 Extended Purse Management Services

The basic services for purse management were described in Section 3.2. There are additional services that a human user may find useful. An example is the "Convert Purse" service which will convert a purse from one payment system to another.

One can argue that this service belongs outside the payment manager for the following reasons:

- It is good practice to keep the design simple,
- This service *can* be provided by an external application based on the services already provided by the payment manager (e.g., deposit, create purse, withdraw etc.),
- A payment through the payment manager should be a *pure* payment. The task of attaching meaning to the payment should be done outside (e.g., by a business application). If the convert purse operation were to be provided by the payment manager, then it has to really interpret the payment and link it to the intended purpose (buy a different kind of electronic value).

On the other hand, one can argue that it must be provided by the payment manager for the following reasons:

- The payment manager is supposed to provide *all* purse management services; it is inconsistent to shift some of them outside, and
- If currency conversion, which is just a special case of purse conversion, is provided by the payment manager, then purse conversion should be as well.

4 Security

As usual, the security requirements fall under three categories: *integrity, confidentiality,* and *availability.* The goal is to meet the security requirements of all the players in the system.

Integrity requirements of a player spell out the expected behaviour of the payment service from the point of view of that player. The bottom-line is that a player should not lose money without her (direct or implicit) approval. Access control is one of the means of implementing integrity requirements. Local access control for the payment manager operations was already described in Section 2.2.4.

Confidentiality requirements of a player spell out the extent to which information *about* the various operations of the system can be revealed to other players. The transfer layer of *SEMPER* will provide a way to specify high-level security requirements for individual transfers (e.g., in the most general case, these requirements will be of the form: anonymity [from specified parties], unlinkability [by specified parties to a speci-

fied degree], message confidentiality). The payment systems may offer some of these services. Each purse has to know which security attributes it can provide for its value transfer operations. This information is obtained from the underlying payment system and its adapter.

The payment manager and the adapters might communicate with remote parties via the transfer and communication managers. The payment systems, being external to *SEMPER*, typically take care of their own communication, even though ideally they should use *SEMPER* transfer services. In fact, in certain situations (e.g., the anonymity provided by ecash can be ineffective if everyone can see the network address of the device that engages in ecash transactions), it may be necessary for the adapter may "intercept" messages sent by the payment system and re-route it via the *SEMPER* transfer service with the requisite security requirements.

5 What Comes Next?

For the beginning we decided to support two payment models

- a cash-like model, with the actual systems "ecash" and possibly "Express" and "CAFE."
- an account-based model, with the actual system "iKP" supporting a post-paid model.

We might add a real cheque system as well as a "home-banking" scenario (which corresponds to Figure 3c and d) later.

Acknowledgements

This work was partially supported by the ACTS Project AC026, *SEMPER*. However, it represents the view of the authors. *SEMPER* is part of the Advanced Communication Technologies and Services (ACTS) research program established by the European Commission, Directorate General XIII.

6 References

BBCS 94 *J.-P. Boly, A. Bosselaers, R. Cramer, R. Michelsen, S. Mjølsnes, F. Muller, T. Pedersen, B. Pfitzmann, P. de Rooij, B. Schoenmakers, M. Schunter, L. Vallée, M. Waidner:* **The ESPRIT Project CAFE - High Security Digital Payment Systems;** ESORICS '94, LNCS 875, Springer-Verlag, Berlin 1994, 217-230.

BGHH 95 *M. Bellare, J. A. Garay, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik, M. Waidner:* **iKP — A Family of Secure Electronic Payment Protocols;** Usenix 1995.

Chau 89 *D. Chaum:* **Privacy Protected Payments — Unconditional Payer and/or Payee Untraceability;** SMART CARD 2000, North-Holland, Amsterdam 1989, 69-93.

JaWa 96 *P. Janson, M. Waidner* **Electronic Payment Systems;** *SEMPER* report 211ZR017, Janurary 30, 1996 (a previous version was published in: SI Informatik / Informatique 3 (1995) 10-15)

Available from <http://semper.zurich.ibm.com/ info/211ZR017.ps.gz>

SET 96 *SET Working Group:* **Secure Electronic Transactions;** Mastercard and Visa, February 12, 1996, Draft 3.2.

Available from <http://www.mastercard.com/set/set.htm>

Waid 96 *M. Waidner:* **Development of a Secure Electronic Marketplace for Europe;** *SEMPER* report 431ZR031, February 19, 1996.

Available from <http://semper.zurich.ibm.com/info/431ZR031.ps.gz>