MeT Wallet Concept Description Version 1.0

Mobile electronic Transactions Ltd.
MeT-Wallet-Concept-v1_0-20030922

Disclaimer:

This document is subject to change without notice.

Contents

1. Scope

The development of mobile commerce is transforming the mobile phone into a trusted repository of commerce-related information - the Personal Trusted Device (PTD). This transformation requires from the PTD new means to satisfy the growing variety of mobile commerce applications in areas such as mobile banking, payment, ticketing and secure access-based services.

MeT addresses those needs by the concept of MeT Wallet - the conceptual place in the PTD through which the user can access information that is related to mobile commerce. At the same time MeT Wallet offers the convenient platform for service providers to deploy a variety of commerce-related services to the satisfaction of the user.

MeT recognises that successful adoption of the MeT framework depends on global acceptance of the PTD by end users. In order to facilitate and encourage mass mobile acceptance, the MeT Wallet platform is needed to encourage and accommodate the consistent deployment of independent services.

This document describes the core concept of the MeT Wallet in terms of architecture, functionality and deployment. Further, it discusses the relationship between the MeT Wallet and other concepts.

2. Document status

The current version of this document is available online at www.mobiletransaction.org.

2.1 Version History

Version	Date	Working Group	Description
0.1	19.06.2003	MeT Concept Group	Initial Version
0.2	29.07.2003	MeT Concept Group	First Revision
0.3	07.08.2003	MeT Concept Group	Changes to reference model
0.4	15.08.2003	MeT Concept Group	Use-cases added
0.5	12.09.2003	MeT Concept Group	Comments from MeT Wallet workshop, example added

2.2 Errata

3. References

OMA provisioning	www.openmobilealliance.org
MeT core specification	www.mobiletransactions.org
MeT CUE	www.mobiletransactions.org
JCP	www.jcp.org
J2ME	http://java.sun.com/j2me/.
MIDP	http://java.sun.com/products/midp/
Symbian	http://www.symbian.com/
BREW	http://www.qualcomm.com/brew/
JSR-177	http://www.jcp.org/en/jsr/detail?id=177

4. Wallet as a concept

Mobile phones are increasingly becoming open platforms. This trend is visible regardless of the geographic market or communication technology, through solutions such as Symbian [Symbian], Java (J2ME) [J2ME, MIDP], and Brew [BREW].

The current deployment of mobile commerce follows two trends. The first trend tentatively accepts the existing mobile terminal with its existing, mostly built-in functionality as it is today but demands additional security. Another trend would like to utilise the growing openness of the mobile terminal to deploy specific mobile commerce applications into the terminal. Again, increased security is required.

This second trend, the combination of increased platform openness and increased demand for security, is the driving force for the concept of the MeT Wallet. The MeT Wallet is the platform where different players may deploy their commerce-specific applications in different configurations while defining and maintaining security level as required by their businesses.

The MeT Wallet draws a lot from the concept that has been successful with smart cards: hiding sensitive data inside the secure environment and making them available in a controlled way through specific functions. At the same time the MeT Wallet demonstrates how such a concept can be leveraged from the closed environment of the smart card to the mobile terminal as a whole, addressing issues such as the multiplicity of environments, user interaction, and service invocation.

The wallet can be seen as a generalised provider of secure services. The secure service is the functionality that encapsulates sensitive data so that they can be accessed only by specific functions. The secure service therefore bears the following properties:

- ability to hide sensitive data and make them indirectly available only through service
- all the sensitive user interaction is handled within the service
- access to services that may can be controlled

The wallet does not define what constitutes the sensitivity, as the sensitivity may be interpreted differently, depending on application area. Instead, the wallet provides a platform that may be used to deploy secure services, regardless of the exact area of applications.

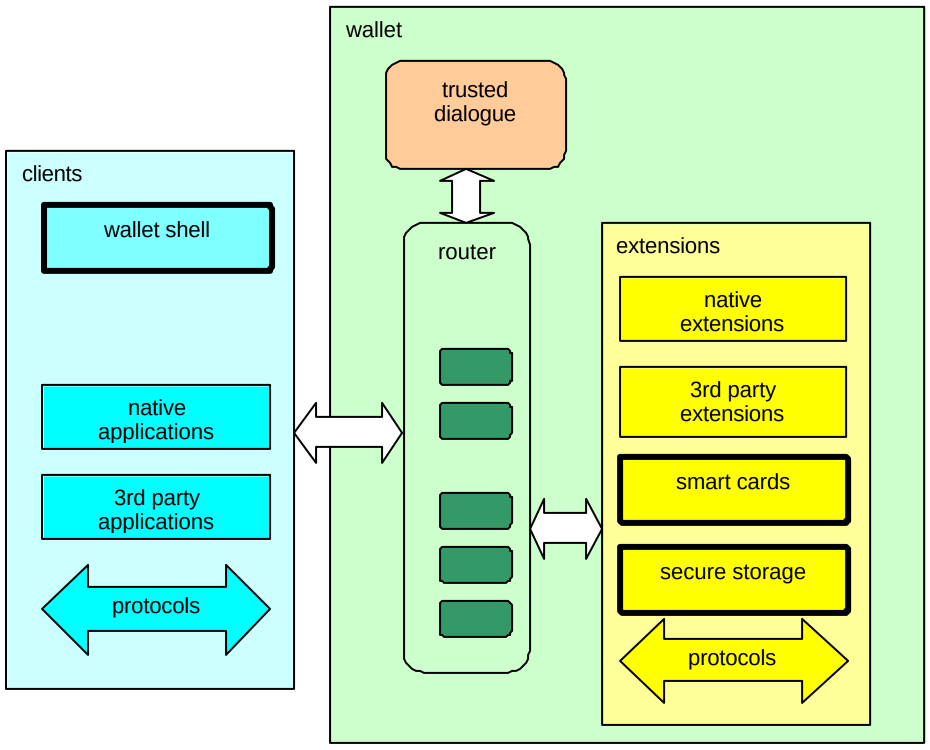
The MeT Wallet is a platform that allows developers to register their applications with the wallet, as wallet extensions. These extensions may be written for different environments.. Each wallet extension may expose several services that may be called by other applications, whether at the same terminal or over the network, or invoked directly by the user. The MeT Wallet provides convenient means for both the user and the application to identify what services from different extensions are available. The extension may control the visibility of its services to applications and is always informed about the identity of the application that is calling the service. Extensions registered with the wallet may further utilise the secure storage and other standard extensions.

The wallet does not enforce all its extensions to behave as secure service providers. The wallet recommends and provides guidelines for developers of extensions to facilitate the process of writing secure services. The wallet expects that several application areas will develop their own practices how to write secure and compatible extensions.

The MeT Wallet has been designed with mobile commerce in mind. However, the generic ability to call services between environments, combined with access control and mutual authentication of the caller and called application may make the wallet attractive to other application areas as well.

5. Reference architecture

The MeT Wallet can be defined in terms of its reference architecture, as depicted below.



The core feature of the wallet is to deliver the ability for application clients to identify and invoke one or more of extensions. Specifically, one of the clients can be the wallet shell that allows the user to directly invoke extensions through the device user interface.

For the purpose of invocation, the wallet provides the router that allows extensions to register with the wallet, to be discovered and securely invoked by clients. Extensions may call other wallet extensions, in particular secure storage and smart cards. They can also use the trusted dialogue.

5.1 Router

The router provides invocation capability to clients, as well as registration and access to trusted dialogue to wallet extensions. The router is the key element of the wallet architecture.

The router works across environments, allowing any client to call any extension, regardless of its actual implementation, even though certain restrictions may apply. The router provides all the necessary interfacing means to translate call methods and data formats between clients and extensions.

The router functionality may not be directly visible to all the clients and extensions, as it is provided through different means, depending on the characteristics of the particular environment.

5.1.1 Registration

The router allows extensions to register with the wallet. The router does not verify or validate the origin of the extension, allowing every properly formed application to register itself with the wallet as an extension.

When the extension is registered, the router creates the registration entry. Such entry stores the reference to the extension as well as all the related registration parameters. The extension itself can be located in any place in the mobile device (e.g. together with other applications of the same type) as long as the router can access and invoke it.

Each extension may register itself several times with the wallet, each time providing a different set of registration-specific data. Every registration causes the router to create the new entry. This allows the extension to handle several objects of the same kind as several registration entries.

The registration should be handled at the time the extension is provisioned. The registration can be also performed dynamically when the extension is being invoked The dynamic registration may depend on the environment of the extension.

It is possible to deregister one or more registration entries of the same extension. If the extension is removed from the system, all its registration entries are removed. Special support for updates of the extension should be defined so that the new version of the extension may inherit all the registrations from the previous one.

The router may implement certain means to automatically discover and register extensions as they become available to the terminal. For example, extensions located on the smart card may be discovered when the card is inserted.

5.1.2 Identification

The wallet provides means for the client to identify what extensions are available to the client for invocation. The exact method available for the client depends on the client environment. In some environments an API may be preferred, while in others integration with existing mechanisms may be preferred.

The MeT Wallet shell plays a special role in the identification process, as it presents extensions directly to the user. In order to facilitate this identification, the extension may provide additional registration information that helps arrange the interface, like informative name, picture or the type of the extension.

As a part of its access control an extension may expose the fact of its existence only to some environments or only to some clients. For example, not all extensions may want to be visible through the wallet shell.

5.1.3 Invocation

The client may call any registered extension. The exact implementation of the invocation method depends on the environment the client is calling from, and it is independent of the environment the extension executes in. It may be possible for the client and the extensions to obtain details of actual environments of their counter-parties.

The invocation is always synchronous, i.e. the call from the client waits for the extension to complete its task. If the client can handle multi-tasking then it can proceed with its own tasks while waiting for the extension. Otherwise the client is suspended.

Whether the particular extension can be re-entered (i.e. called more than once at the same time) depends on capabilities of the environment and properties of the extension. The wallet neither requires nor assumes re-entrancy, but allows extensions to utilise re-entrancy if they support it. The wallet does not currently define the relationship between the invocation and the instantiation of the extension.

The wallet does not implement callback methods (where the extension calls the client), but such methods may be used as a part of the actual interface at the client or extension side. Similarly the wallet does not define any specific method that allows an extension to pass large amount of data back to the client. Some extensions may utilise environment-specific messaging, file sharing or similar techniques, using the wallet invocation to pass the reference to such data.

5.1.4 Access control

The router provides access control to identify both parties and then selectively allow client to access extensions.

The router identifies both parties (the client and the extension), using techniques dependent on the environment. The router does not identity the extension or the client by itself, but it relies on the environment to perform the identification for the wallet. For example, Java MIDlet may be identified by its trusted domain, as defined by MIDP2.0 while the remote client may be identified by the certificate used by the protocol to secure the connection.

Each extension may publish its access control preferences that list which clients from what environments are allowed to discover and/or invoke the extension. For example, extensions may prefer not to be available from the wallet shell or they may accept invocations only from clients provided by affiliated business partners. These access control preferences of the extensions are used by the router and will not be available to the clients.

During the identification process or latest during the invocation the router may exchange identification information of the client and the extension so that both parties are informed of the identity of the other party. This information can be used by either party to process or decline further information exchange.

The concept of access control assumes that all parties trust in the platform (mobile terminal) to properly implement access control. The trust seems to be justified as parties already trust the platform to properly implement environments, communication protocols etc.

Access control may be perceived differently in different environments, depending on capabilities of the given environment. Specifically, the extent the access control is available may differ between environments. The MeT Wallet defines minimum requirements for each environment regarding access control.

5.2 Wallet extensions

Wallet extensions deliver functionality that can be invoked by the client. The most important role of an extension is to deliver a wealth of services to the user in a secure manner. The extension may hold confidential data, which is not disclosed directly to clients. In this respect, the extension acts as a large 'smart card', guarding access to its internal data and providing limited access through selected APIs.

An extension may for example implement a stored value payment system. The current balance may be stored in the secure storage (which is one of the standard extensions). The balance can be delivered by the extension directly to the user when called from any client (including the wallet shell), but payment and reload operations may require clients that are known to the extension or clients that are able to communicate with the central server. No other access methods are allowed.

An extension may be embedded in the terminal's basic software ('native' extension) or it may be downloadable from third-party developers. An extension may reside on the terminal memory, in a smart card, in a removable storage media or on a remote server, securely accessible through MeT Wallet protocols.

Wallet extension may be implemented in a different environment than the client, with invocation through the router. For instance the extension may be called directly from the wallet shell.

The extension itself is usually an application, with all the relevant rights to use whatever mechanisms are available in the given environment. For example, an extension that is implemented as Java MIDlet may access all the mechanisms that are normally available for MIDlets. Specifically, an extension may communicate with the user, may exchange information with remote servers or it may use other elements of the terminal hardware or software. The MeT Wallet does not preclude the extension from using any of those features, nor does it impose any special restrictions.

Extensions may access smart cards if appropriate mechanisms are provided by respective environments. The role of smart cards is discussed in details later in this document.

Extensions can act as clients and can also identify and invoke other extensions. In particular, an extension can always call standard extensions: secure storage and smart cards. The ability to call an arbitrary extension (not the standard one) may be restricted by limitations of the actual implementation. Some implementations may offer certain means to address such limitations. For example, in implementations not supporting multi-tasking capability, chaining of extensions may be handled by suspending the calling extension and running the called extension. Implementations having no such restriction can run extensions in parallel as allowed by their implementation.

5.3 Clients

MeT Wallet router aids MeT Wallet clients to invoke MeT Wallet extensions.

Clients can be of different kinds. They may be native applications, embedded in the terminal software or they may be downloaded applications (e.g. Java). Wallet Shell and protocols (protocol clients) can also act as clients to the wallet directly invoking extensions.

One client may call several extensions, depending on needs. The wallet does not mandate any particular dynamics for invocation, but client environments and limitation of the implementations may impose certain restrictions.

The client is an application operating in its environments and enjoys all the rights to use its environment just as any other applications in the same environment. For example, the client may communicate with the user or establish its own communication link with the remote server. The wallet does not preclude or restrict any of those functions.

5.3.1 Wallet Shell

The wallet shell is one of the wallet clients and - similar to other clients - it can identify and invoke extensions. The most important difference is that the main purpose of the shell is to present the list of extensions directly to the user.

The wallet architecture does not require the existence of the wallet shell nor does it limit the number of actual wallet shells. There may be several client applications (e.g. MIDlets) that operate as wallet shells.

Access control is designed in such a manner that every extension may decide whether some of its registration entries can be visible through the wallet shell. Further, each registration entry may limit its visibility to some clients only, so that some implementations of the shell may not have access to all extensions.

It is expected that there will be at least one default shell always available to the user so that the user will be able to enjoy the convenience of directly invoking extensions.

5.4 Special extensions - secure storage, smart card access

The wallet provides some extensions that are designed primarily to be used by the other MeT Wallet extensions to improve their functionality. In particular, the wallet provides access to secure storage and access to smart cards.

The exact method that is used to access these special extensions depends on the environment. The wallet does not mandate any specific access or implementation, but assumes that each environment will eventually provide access to special extensions, if they are available.

In some environments those extensions may not be called through the wallet API, but they may rather expose their own APIs or integrate into other APIs. Therefore they may be perceived more like utility libraries than extensions. They are however considered as part of the wallet architecture.

Note that as they are extensions, they can be called either by another extension or directly by the client. The wallet does not preclude either possibility.

5.4.1 Secure storage

Secure storage is the storage within the terminal that holds information in an encrypted form. Access to the storage is protected, usually with PIN. The storage implements two basic services: store and retrieve. In order to receive information from the storage in a plain format, the user must supply the valid PIN.

Terminal may use several methods to implement the secure storage, depending on expected security level, available support from the hardware and similar. The wallet defines minimum requirements for the secure storage.

The secure storage can be made available to clients and extensions by different means. For example, in the Java MIDP environment, secure storage may be implemented as a separate API or as an extension to the existing RMS system.

Detailed discussion of the secure storage is provided later in this document.

5.4.2 Smart card access

Smart cards are tamper-resistant devices that can be fitted in the terminal. Each smart card contains a microprocessor and may host one or more applications. The most popular type of smart cards in the GSM environment is the SIM card that authenticates the subscriber for billing purposes and stores additional information (phone book, provisioning etc).

The extension can access smart card in order to execute applications on the card and to manipulate data on the card through such application. One smart card may possibly have several applications, ranging from the passive data store to active support for security functions to implementing proactive applications. The wallet interprets each application on the card as the separate extension.

Cards that do not conform to the conceptual model of the wallet cannot be interpreted as extensions. This does not preclude the existence of certain library methods to operate on such cards, but such libraries are not within the scope of the wallet architecture.

5.5 Trusted dialogue

In addition to the wallet shell (that acts as a client), the wallet provides the collection of functions that can be invoked by extensions for the purpose of trusted dialogue with the user. These methods are available for extensions to use in addition to methods that can be normally used by them to communicate with the user. Extensions have no obligation to use methods provided by the wallet.

The trusted dialogue offered by the wallet serves two purposes.

- Provide consistent user experience. Trusted dialogue methods are constructed to follow expected functions rather than to provide generic input/output capabilities. As such, the user may experience the similar interaction regardless of the functionality of the actual extension. As the wallet operates across different environments, extensions may provide similar user experience regardless of their environments.

- Support trusted interaction. The wallet is able to determine the identity of the extension and it can communicate such identity to the user in a manner that cannot be imitated by any other software. By this, the wallet may reassure the user about the genuine origin of the party that initiates the dialogue.

Note that clients can use the trusted dialogue in order to promote the consistency of user experience. The identification of the caller by the trusted dialogue is supported to the extent the caller can be identified by the wallet.

6. Example

The following example is used to illustrate how different aspects of the wallet functionality can be expressed in the assumed wallet architecture. The example stresses technical side of the wallet and should not be interpreted as a use scenario. The narrative is used only to illustrate the possible usage.

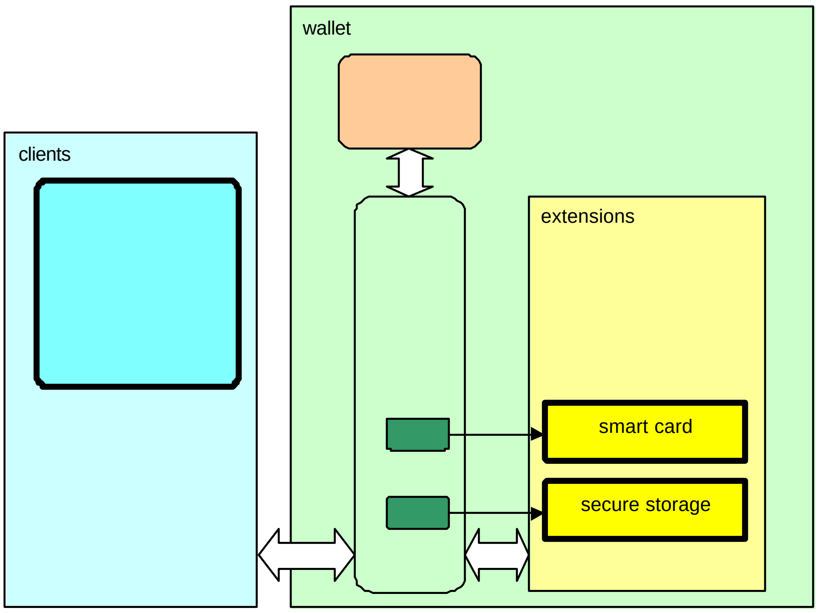
An example is constructed around the mobile ticketing application area, where tickets are acquired over the air through the remote connection and then stored in the mobile device.

6.1 Initial state

The initial state of the wallet is as follows. There are two standard extensions registered with the wallet. The smart card extension allows for an access to the contact or contactless card. The secure storage provides place to store sensitive information, encrypted and PIN-protected. The wallet shell is the only client present.

The smart card extension and the secure storage extension can be accessed by other clients, possibly with certain restrictions. They cannot be accessed from the wallet shell so that the shell is empty.

The smart card extension may hold different applications e.g. applications that implement contactless train tickets. Initially the smart card does not have any applications of this kind.

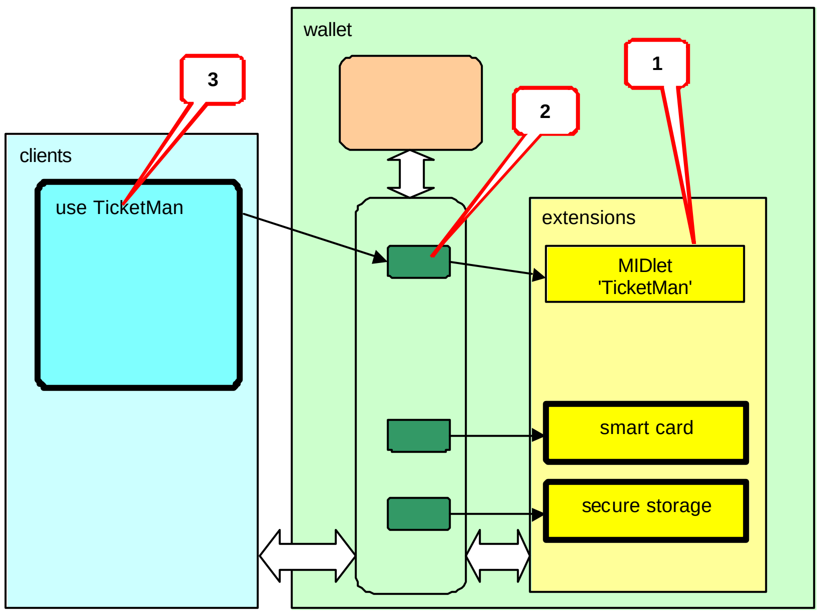


6.2 MIDlet Extension Loaded

This example demonstrates how the loaded extension can integrate with the wallet as one of wallet extensions and then how it may become available to the user.

The MIDlet 'TicketMan' has been developed to operate as the wallet extension. The purpose of this MIDlet is to allow the user to buy event tickets remotely over wireless Internet. Tickets will be securely stored in the mobile device. The user should be able to acquire tickets, manage them (view, remove etc) and finally use them with the help of this MIDlet.

The MIDlet extension is designed in such way that it can accept different start-up parameters that can be passed to the extension by the caller. Also parameters that are stored in the router's entry will be passed to the MIDlet when it is being invoked.



Following is the possible scenario how the MIDlet can be integrated with the wallet. Numbers before paragraphs refer to the picture above.

1. The MIDlet arrives to the mobile device. The exact method to download the MIDlet is not considered here, but at certain moment the downloading process discovers that the MIDlet is a wallet extension that would like to be accessible through the wallet shell under the name 'use TicketMan'. The MIDlet may provide additional registration parameters.

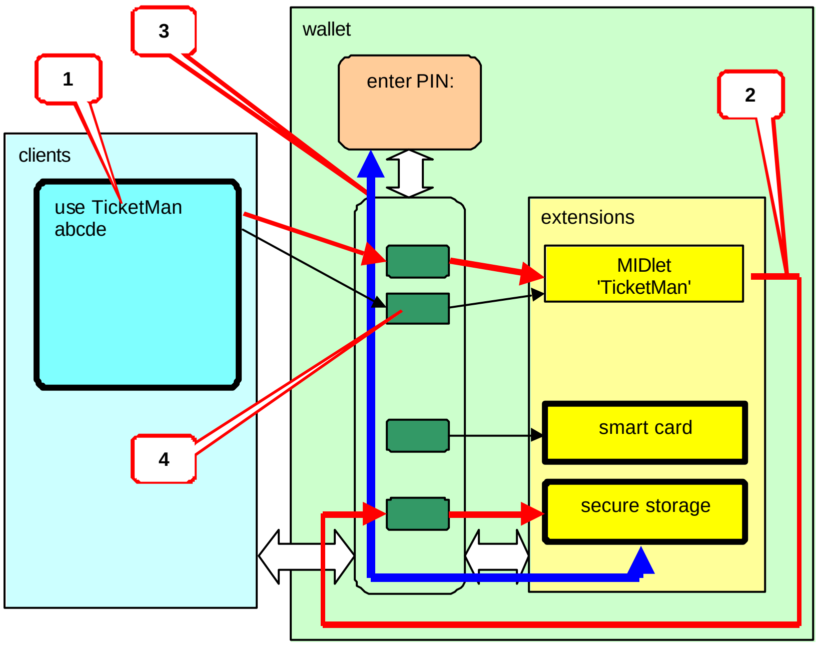
2. The downloading process registers the MIDlet with the wallet by placing the reference to the newly loaded MIDlet in the wallet's router, thus creating the new entry. The entry includes the name as well as access rights that allow the wallet shell to identify and invoke the MIDlet. Other registration parameters are stored in the entry as well. The MIDlet itself can be located wherever in the mobile device (e.g. together with other MIDlets), it is the reference that determines that the given MIDlet is an extension.

3. The wallet shell determines that there is a new extension that would like to be accessible from the shell. The exact method used by the shell to determine changes in the router is not discussed here. The shell retrieves information from the router and creates the entry that is visible to the user.

6.3 MIDlet Extension in Action

This example demonstrates how the MIDlet can use wallet features. The user invokes the MIDlet in order to buy the ticket. Exact methods used to find the ticket and pay for it are not considered here.

The ticket is received by the MIDlet from the remote server and should be securely stored in the mobile device. For that purpose the MIDlet is supposed to use the secure storage server. The secure storage is protected by PIN that must be supplied by the user. In order to increase user's convenience, the new ticket should be accessible for the user directly from the wallet shell.



Following is the possible scenario how the MIDlet can be used to acquire and store the ticket. Paragraph numbers refer to the picture above.

1. The user selects an entry from the wallet shell. The shell invokes the MIDlet through the wallet's router. Note that the MIDlet is invoked with the same set of data that has been passed to the router during the registration. The MIDlet is therefore aware that it has been called using the entry that has been associated with the shell entry 'use TicketMan'.

2. The MIDlet acquires the ticket (e.g. the movie ticket) from the remote Internet site and places it into the secure storage. For that purpose the MIDlet uses the wallet to invoke the secure storage extension. The MIDlet acts at this moment as wallets' client.

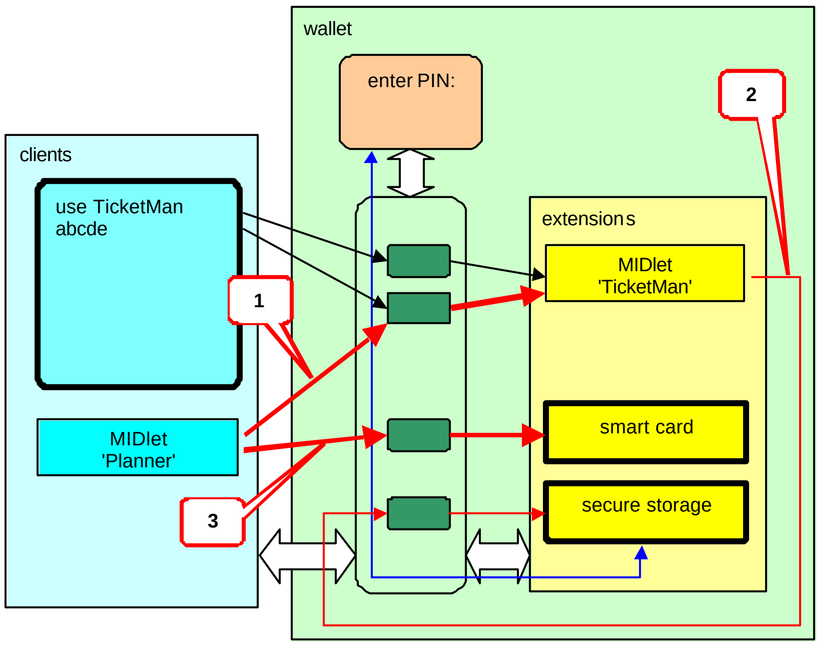
3. The secure storage is encrypted and can be opened with PIN. The secure storage invokes the trusted dialogue through the router. The user enters PIN that is used by the secure storage to successfully open the storage and store the ticket. Note that the PIN is not supplied by the MIDlet.

4. The MIDlet registers itself with the wallet again, this time providing the name of the ticket (e.g. 'abcde') and possibly the reference to the storage area where the ticket has been stored. The wallet creates the new registration entry. The wallet shell puts the name of the movie on the user interface. If the user wants to operate on this ticket it is enough to select it from the wallet shell. In addition, this entry can be also accessed by other clients.

6.4 MIDlet as a Client

A separate MIDlet (named 'Planner') has been loaded in the mobile device. This MIDlet has been developed to take care of time planning. Its main function is to interface with the device's PIM system. In order to fulfil those functions the MIDlet collects different information from various elements of the mobile device. This includes access to tickets stored in the mobile device.

The Planner does not understand format of tickets that are processed by TicketMan, but it is able to read tickets if they are presented in the standard format (e.g. in MeT format). The TicketMan can provide such information if invoked with certain parameters.



Following is the possible scenario how the MIDlet client can use the wallet. Paragraph numbers refer to the picture above. It is assumed that the client MIDlet Planner has been already loaded to the mobile device.

1. Once started, Planner checks with the wallet whether there are any extensions of interest that it can access. The entry of 'TicketMan' that holds the ticket is spotted and the Planner invokes the entry in order to retrieve ticket details. Note, that Planner does not have access to the first entry associated with TicketMan.

2. The TicketMan proceeds with the ticket as shown in the previous example (accesses the secure storage, the secure storage asks for PIN etc.), as shown by the thin dotted line. The Planner receives the necessary ticket details in standard format.

3. The Planner accesses the smart card extension to check whether there are some train tickets. If there are no tickets, the Planner does not receive any information.

7. Use cases

The following is a description of some selected use cases. These use-cases demonstrate the expected impact of the wallet on the market of secure services. They also illustrate the architecture of the wallet and highlight the most important technical solution used within the wallet.

7.1 Terminal-based tickets

This use case describes the system where the terminal is used to process event tickets. Such tickets can be stored directly in the terminal, e.g. in the secure storage provided of the wallet as one of its standard secure services.

The use case also illustrates how the service that is using the wallet can be deployed on non-wallet terminals, thus demonstrating how to bridge the initial gap of insufficient installed base of wallet-enabled terminals.

7.1.1 Description

- Service provider wants to establish the new service to sell event tickets. Tickets will be stored in terminal memory, e.g. in the secure storage provided by the wallet as one of secure services.

- Service provider creates MIDlet that can handle tickets. The MIDlet provides the complete coverage for tickets from the given service provider, i.e. tickets can be bought, viewed and used with the help of the same MIDlet. Such MIDlet can be loaded from the Web or it can be distributed by any other established method.

- The MIDlet is able to detect whether the terminal provides the wallet. If not, the MIDlet can operate without the wallet, even though the usability may degrade. The minimum requirement for the terminal is the regular MIDP environment.

- When the MIDlet is loaded to the terminal it requests to be registered to the wallet shell with the name that suggests to the user to buy new ticket from service provider. The MIDlet therefore establishes itself as a secure service. If the terminal does not support wallet registration, the request is ignored and the MIDlet is stored together with other MIDlets or applications, under the name that identifies the service provider.

- As the MIDlet is registered to the wallet shell, the user can invoke the MIDlet to buy the ticket. The MIDlet can handle the entire shopping experience. The actual payment method is out of scope of this use case, but it can also be handled by the MIDlet, possibly using other information from the secure storage (e.g. credit card information).

- Tickets that are acquired from the service provider are stored by the MIDlet to the secure storage so that they cannot be used without the proper user authentication. If no secure storage is provided, tickets can be stored in the regular storage, even though it can provide significantly less protection. Further, the MIDlet may use the wallet trusted-dialogue to query the user for PIN and establish certain encryption by itself.

- All tickets that are handled by this MIDlet are also listed on the wallet shell, pointing to the same MIDlet but providing different start-up parameters for the MIDlet. Whenever the user selects the ticket from the wallet shell the MIDlet is started to handle the user's request regarding this ticket like viewing, use, removal, transfer etc. If the terminal does not support the wallet registration, all tickets can be handled by starting the MIDlet the normal way.

7.1.2 Technology required

This use case requires the Java environment in the terminal. The use case is also applicable for other programmable environments if they support the wallet.

This use case requires the following wallet-related technologies

- shell registration
- secure storage service
- optionally: the trusted dialogue

This use case demonstrates also how the application can be deployed without the wallet functionality available in the terminal.

7.1.3 Benefits

For the service provider this use case demonstrates the following benefits.

- *The use of established technology (Java).* Initial investment is minimal as well established tools and method can be immediately taken into use. This minimises both the cost and the risk of entering the market.

- *Backward compatibility*. The use case demonstrates how the same MIDlet can operate on terminal with wallet functionality and on terminals without them. In the latter case the usability and security degrades, but the ability to address such terminal allows the MIDlet to reach the very large installed base of customers.

- *Visibility on wallet shell*. The ticketing MIDlet can present itself to the user on the wallet shell that is dedicated for commerce-related items, so that the user can easily find it when needed. This may improve the eagerness of the user to make use of such application.

- *Improved data security*. Ticketing information is PIN protected in the secure storage. If the terminal is lost or stolen, tickets cannot be used by an unauthorized person.

- *Flexibility in implementation*. The service provider is not tied by any specific protocol or data format. The ticketing MIDlet can interact with the established infrastructure. It can also interact with the user in a manner that the service provider finds most appropriate.

- *Branding ability*. Both the MIDlet itself and the wallet shell can deliver brand to the user. Names, pictures or other branding elements can be used to provide association of the brand and the service.

For the user this use case demonstrates the following benefits.

- *Consistent usage experience.* The integration of all the commerce-related items into one menu area allows for more consistent navigation. Elements of user interaction that are provided by the secure storage or through the wallet trusted dialogues are consistent.

- *Usability*. As the ticketing application is reachable from the wallet shell that is specifically designed for commerce-related items, the user can more easily find the relevant MIDlet. Further, all the tickets that have been already acquired can be accessed easily from the same wallet shell.

7.1.4 Market impact

This use case is aimed at early stages of the development of the market for the wallet-based solutions. As such, it concentrates on issues that are characteristic to transition time, namely the co-existence of wallet and non-wallet devices.

- *Immediate appeal*. The use case demonstrates how the initial gap of a missing installed base can be overcome with the help of MIDlets that can optionally take advantage of the wallet functionality while still delivering certain functionality on non-wallet devices. The market impact of backward compatibility is enormous as it drives the gradual adoption of the wallet while satisfying the needs of service providers for a large installed base.

- *Gradual adoption*. The single MIDlet that acts as a provider of secure services is again characteristic for the early market where individual providers can deploy their solutions in relative isolation, still enjoying immediate benefits coming from the usage of the wallet. The wallet does not require any specific market configuration, thus enabling gradual market adoption without any need for specific business alliances.

- *Higher value*. The usage of secure storage for tickets helps providers to remove certain fears regarding the security of data stored in the terminal. The availability of secure storage can therefore drive the market from low-value goods towards the processing of higher-value items, e.g. tickets.

7.2 Banking identification

This use case shows a bank deploying a handler to provide a secure identification service. This service can serve not only the bank's own applications (whether Java MIDlets or web pages) but it can also be used by some of the bank commercial partners, if authorised by the bank.

The banking identification usage case demonstrates how the single player can create the affinity structure around its wallet services if only the wallet is used as a foundation for such services. This use case demonstrates the growing sophistication of solutions, enabled by the wallet architecture, that allows for new revenue possibilities.

7.2.1 Description

- Bank deploys the identification system that bases e.g. on non-standard cryptographic features, e.g. on the list of one-time passwords. Such identification system is specific to the bank, i.e. it is only the bank server that can verify the actual identity.

- The system is deployed as the MIDlet handler that uses the wallet secure storage to keep the list of passwords. Once the list of passwords becomes empty, or at the convenience of the user, the handler communicates with the bank to receive the new list. This communication requires the user to provide additional identification (e.g. password) that can be delivered off-line.

- The handler provides at least one secure service: it discloses one password at a time to approved clients. Other services, if any, are not important for this use case.

- Clients are restricted to be MIDlets signed by certain providers (specifically the bank itself) or web pages, if accessed over secure connection with approved certificate (again, specifically the banks own). The handler is delivered with the initial list of client identities that can use the handler. The handler can be also activated from the wallet shell to start the re-load of the password list.

- The bank may allow its partners to use the one-time password method for their services. Each partner is then allowed to deploy the MIDlet that can access the handler. Alternatively, each partner can create the web page that can be accessed through the secure connection. In both cases the cryptographic identity of the client must match the access control established by the handler.

- Clients coming from bank's partners (e.g. affiliated stock trader, insurers, on-line shops etc), be it MIDlets or web pages, can use the banking identification server to provide identification for their own services. They can call the handler, receive one-time password and then use it to authenticate the client through the banking server.

- The bank may use the communication intended to update the list of passwords to modify the access control list, thus allowing new partners in or excluding some of existing partners, thus shaping the market for its identification servers.

- Note that there are several alternatives not discussed here that may allow the bank to gradually position itself on the market. For example, the strategy of dual-use MIDlets presented in the first use case is also applicable here. The service of identification can be bundled together with certain banking services (e.g. balance query) into one handler. Such handler on on-wallet terminal can be only used to access banking services while the same MIDlet deployed on the wallet-enabled terminals can be used as the foundation for affinity services as described above.

7.2.2 Technology required

This use case requires the Java environment in the terminal. Optionally, the browser environment may be used as well. The use case is also applicable for other programmable environments if they support the wallet.

This use case requires the following wallet-related technologies

- router with MIDlet to MIDlet invocation (optionally with browser to MIDlet invocation)
- secure storage service
- wallet shell registration (optional)

7.2.3 Benefits

For the bank this use case demonstrates the following benefits.

- *The use of established technology (Java).* Initial investment is minimal as well-established tools and method can be immediately taken into use. The solution can be supported by existing developers and software products.

- *Terminal-based security*. The solution does not require any hardware-specific security to be installed in the terminal or delivered to customers; the complete solution fits the regular terminal (with the wallet). This is not only cost-effective, but it also addresses the large base of terminals.

- *Infrastructure leverage and re-use*. An infrastructure that is already in use to support existing authentication method is likely to be re-used. The traditional authentication method (e.g. one-time password list on paper) can be still used during the transitional period.

- *Market reach limited only by wallet deployment*. As no additional hardware devices are needed (not even a card), every wallet-enabled terminal can be potentially used. Therefore the potential market is quite large. In addition, the logistics of the deployment is significantly simplified and is cost-effective.

- *Branding opportunity*. The handler itself can communicate the brand not only through the wallet shell but also during every interaction when queried by the client. Not only it increases brand awareness but also improves information security and transaction trust as the user becomes aware of the bank involvement in transaction.

- *Market creation*. The wallet allows for the creation of the market for secure services where the bank provides the service and then benefits from re-selling the service to other players. The handler can additionally serve audit purposes if the server-based audit is not sufficient.

For the user this use case demonstrates the following benefits.

- *Consistency of identification*. Different clients use the same handler to secure the user's identification. From the user perspective it is always the same, consistently presented interaction that leads to authentication. The solution allows the user to create the consistent expectation regarding the identification, making it easier and more understandable.

- *Increased trust generated by brand*. As the brand of the bank can be displayed for every identification, the user's trust in transaction itself may increase due to the fact that the bank indirectly supports the activity of the client. This in turn may increase the willingness for the user to engage in transactions.

7.2.4 Market impact

This usage case illustrates more mature market of secure services. As such, it concentrates on the ability to create new markets by allowing several clients to use the same handler.

- *New business model*. The use case demonstrates the new business model where the handler originally intended to satisfy needs of the individual provider can be re-used by other providers. This creates new business opportunities for the original provider (the bank) as well as for subsequent service providers which can access the increasing installed base.

- *Synergetic development*. Even though the use case requires certain provider (the bank) to make the first step and issue the handler, it is really the synergetic development of the market that happens. The bank, through its brand, builds the trust while affiliates build the necessary market momentum.

- *User willingness to participate*. The mobile commerce has been plagued by low user interest. The use case demonstrates how the user can be attracted into using the handler by the growing number of offerings that are coming from the bank and from affiliated companies.

7.3 Smart card stored value payment

The use case shows the operator that deploys the stored value payment system that bases on SIM through secure services of the wallet. The main purpose of such service is to be used by the variety of applications (MIDlets) for small payment purposes.

This use case demonstrates how the operator can build new services by leveraging its position in the terminal (ownership of tamper-resistant SIM) without allowing unrestricted access and endangering the sensitive content of SIM.

7.3.1 Description

- Operator deploys the stored value micro-payment that is aimed initially at Java gaming. The value is stored in SIM in a format that is operator proprietary. The operator can adjust the balance in SIM through any of the established methods (e.g. by sending encrypted SMS).

- It is out of scope of this use case how the value is established, but such value may come e.g. from the pre-paid user's account or it may be actually credited to the user's account. A similar use case can be constructed for other means of payment where SIM is used rather as an identifier than as value holder.

- The operator deploys the handler (presumably the MIDlet) signed by the operator that provides the ‘pay’ service to clients. The handler is registered to the wallet and uses some smart card access methods (e.g. SATSA/JSR177) to access SIM. The handler is further registered to the wallet shell so that the user can access the account for management purposes (e.g. balance query).

- Clients of such handler are all MIDlets that may require payment (presumably of small value) during their operations. For example, a game may require payment before it allows the user into the next level. Similarly, multimedia manager may require a small fee before allowing the user to play the specific song.

- Whenever the MIDlet requests a payment, it invokes the respective service from the handler through the wallet. The handler contacts SIM and deducts the required account. The user is consulted about the exact value of payment. The user must approve the transaction. The client can be presented with the proof of payment, e.g. in a form of signed document that can be verified with the operator's certificate.

- Access to the service is limited to certain MIDlets that are identified e.g. by the certificate of the signer. The handler arrives with the initial list of approved clients, but such list can be modified by an operator at any time e.g. by sending the SMS to the handler.

7.3.2 Technology required

This use case requires the Java environment in the terminal. The use case is also applicable for other programmable environments if they support the wallet.

This use case requires the following wallet-related technologies

- router with MIDlet to MIDlet invocation
- access to smart cards
- wallet shell registration (optional)

7.3.3 Benefits

For the operator this use case demonstrates the following benefits.

- *Established technology (Java)*. The operator can deploy the solution to the variety of terminals (assuming they support the wallet concept), using tools and resources that are already present on the market.
- *The use of SIM for additional services*. As SIM is changing into USIM with place for several applications, operators are facing the challenge to capitalise on this development. The wallet allows them to exploit additional applications on SIM without surrendering its total ownership.
- *Leverage of pre/post paid relationship*. The demonstrated payment system can be easily integrated with operator's pre-paid or post-paid relationship, thus leveraging them. It will result in the increased revenue per user.
- *Market reach limited only by wallet deployment (no extra hardware)*. As the solution does not require any additional hardware (the wallet and SIM are the only prerequisites), the potential market size is quite large, minimizing the risk of investment.
- *Branding ability*. The operator can establish itself as the preferred payment brand that is visible during every transaction. The use of Java-based handler means that the brand (as well as the user interaction) can be presented to the user in a rich form.
- *New role and new market potential*. Re-selling SIM-based services opens new market possibilities to the operator as it is assuming a new role. Several business models can be implemented using the same technology.

For the user this use case demonstrates the following benefits.

- *Simple account management*. All the small payments can be conveniently tied to the single account. This allows the user to simplify the payment relationship management. At the same time the selection of pre-paid or post-paid allows the user to better control accounts, e.g. in case where the corporate phone is used for private purchases.
- *Existing payment relationship*. There is no need to go beyond existing payment relationships, but the scheme utilises the relationship that is already in place and is already tied to SIM. An update of the terminal does not terminate the relationship, assuming that the new terminal supports the wallet.
- *Increased use of digital goods*. Convenient payment is one of the elements that increase the use of digital content. With prices for the content as low as 0.01EUR (one-time streaming), the convenient low-value payment can become effectively an enabler for certain types of consumption.

7.3.4 Market impact

- *Energise mobile commerce*. The convenient payment contributes to the overall increase of the volume of mobile commerce, eliminating additional accounts and decreasing transaction costs. Further, as the user is in continuous control over the standing of his/her account, the user is more willing to venture into new forms of media usage (e.g. streaming, rental etc) and into new forms of payment relationship (e.g. pay as you go).
-

- *Leveraging existing SIM*. Several technical problems stall the usage of SIM for more than the subscriber identifications, thus decreasing the usage of the popular tamper-resistant platform. The wallet allows the SIM to securely and selectively export its features to third-party applications without significant investment from operators (e.g. without updating to USIM). This value proposition is novel in the market and may create several new opportunities.

- *Enabler for third party services*. Third parties had problems capitalising on the investment in attractive content or services as the delivery channel (and specifically the payment channel) has been monopolised by operators and has been subject to crude business models. The wallet allows the establishment of more fine-grained relationships that provide more flexibility to third parties in how and when to charge the user. Effectively, third-party providers face new incentives to deliver attractive content.

7.4 Receipts management

This use case illustrates a retailer that deploys the digital receipt payment. As the acceptance of digital receipts grow, the integrated application can take care of receipts that are coming from different, possibly incompatible sources, to present the user with the integrated receipt handling system.

This use case demonstrates how the standard approach to the client interface allows to create the market for integrated applications while allowing each individual player to retain exclusive control over their data and data format.

7.4.1 Description

- Certain on-line store deploys the MIDlet handler that manages digital receipts. Whenever the user buys goods from the shop, the store activates the handler that receives and stores the receipt. For example, the handler can be registered as a MIME handler for the data content of type 'receipt', so that whenever the browser receives the receipt, it is passed to the handler. Such MIME type registration is currently discussed by JSR211. Even though MIME registration is not directly applicable to the wallet registration, both methods are compatible so that one MIDlet can serve both as MIME handler and wallet handler.

- The handler is registered with the wallet and is accessible through the wallet shell. The user can for example view receipts stored by the handler, remove them or possibly send to other location(s). In addition, other clients can access the handler if they come from the same store. The handler exports the receipts to the client in a standardised manner, regardless of the actual format of the receipt used by the store.

- Some affiliated shops utilise the same concept of MIME type and consequently use the same handler. Other shops, however, use different formats for digital receipts and different delivery means. Such shops may deploy their own handlers for receipts that may be registered with the wallet, talking benefit of being reachable from the wallet shell. All the handlers can be reached also by designated clients. All handlers may store receipts in a manner that is most convenient for the issuer of such handler, but they present the standard receipt format when they are called by the client.

- The original store deploys the MIDlet client that manages receipts e.g. by integrating them into personal finance management system. This MIDlet acts as a client that initially contacts the handler that has been originally deployed by the store.

- The store agrees with other shops so that the client from the original store can access other handlers. Despite the fact that the internal formats of receipts coming from different shops may be incompatible, the user can access all the receipts through one client MIDlet, thanks to the standard interface between the client and the handler. The issuer of each receipt retains all the rights regarding the receipt policy (e.g. cryptographic protection, life span etc.).

7.4.2 Technology required

This use case requires the Java environment in the terminal. The use case is also applicable for other programmable environments if they support the wallet.

This use case requires the following wallet-related technologies

- router with MIDlet to MIDlet invocation
- wallet shell registration

7.4.3 Benefits

For the shop this use case demonstrates the following benefits.

- *Established technology (Java).* The retailer can use the technology that is popular so that it benefits from the widespread understanding of the technology among developers as well as from access to all the tools.
- *Control over own receipts*. The retailer is able to dictate access rules for its own receipts as well as the receipt structure that fits its actual infrastructure. Security concerns as well as additional information needed for the shop can be addressed.
- *Possibility for individual deployment, depending on needs*. The solution can be deployed by each shop or each chain individually, without the need for any market-wide synchronisation of activities. Each shop can choose the deployment route that best sits its individual needs.
- *Market reach limited only by wallet deployment branding ability.* The solution does not require any special equipment in the terminal (except for the wallet itself). Hence the potential market is not limited to the particular model or particular accessory, but is as large as the wallet.
- *Extended reach to affiliated partners*. Once the solution is deployed, it can be easily utilised by partners. The owner of the solution can extend the affiliation by controlling access to the handler, according to business needs.
- *Opportunities for new applications*. New applications can be built on top of the solution. Not only receipt manager (presented in the use case) but also promotional coupons, advertisements etc. can be added to the solution.

For the user this use case demonstrates the following benefits.

- *Convenience of digital receipts*. The existence of digital receipts removes unnecessary paper from the user's wallets and allows for smooth integration with a variety of systems, both at home and in the office (e.g. for business travellers).
- *Consistency of experience*. The variety of access styles proposed by individual shops can be replaced by more consistent experience provided by an individual client. As such, the user gains better control and understanding of the operation of receipts.

7.4.4 Market impact

- *Market creation*. The use case shows how the market for digital receipts can be built from scattered solutions that serves one shop only towards the solution that can provide the critical mass sufficient to drive receipts into market reality.
- *Consolidation*. Receipt-handing systems are currently proprietary to different shops (or chains) that does not allow for the receipts market to reach sufficient consolidation that will justify the introduction of new technology. The wallet concept demonstrates how this can be overcome.

- *Individual control*. Even though the solution allows for consolidation, individual players retain control over their respective data. This allows them to overcome initial fears of the cooperation. It also allows them to address needs of their respective infrastructures (e.g. in terms of additional data fields or required security) without hampering the interoperability.

- *Improvements in logistics*. The arrival of digital receipts allows for significant savings at the point of sales, both in terms of material and time needed to print paper receipts. Several sales channels may be integrated so that e.g. on-line shops that also operate through physical ones can share the same format of receipts, allowing for more flexible pick-up and returns.

- *Business opportunities*. Integration of various receipt formats creates new business opportunities in the form of integration packages. Ability to present the brand as well as some advertisements and coupons creates new ways to leverage the brand and promote new products.

8. Accessing the wallet

The wallet allows access to extensions from several environments. An 'environment' is understood here not only as the traditional programmatic environment (e.g. Java , Symbian), but also as any other type of access environment that may exist in the terminal (e.g. wallet shell, remote access through e.g. browser etc.).

The following is a short discussion of how the functionality provided by wallet extensions can be accessed from some environments.

8.1 Programmatic environments

Programmatic environments include all the environments that allow the application to be deployed into the terminal. Examples are: Java MIDP environment, Symbian environment or even terminal specific ('native') programming environment if it can accept new applications.

In programmatic environments wallet extensions are accessible through some form of invoke-response means, i.e. the client application invokes the extension, typically by issuing a call to the specified function and the extension responds with return information. Depending on the environment this generic call-response method may take on different forms, be it API, the use of URL, internal messaging, remote invocation similar to RMI or Corba or the use of the specific feature of the programming language itself.

The wallet itself does not require any specific access method but it adopts to the method (or methods) that is most beneficial in the selected environment as long as the method satisfies the requirements of the wallet. It can be assumed at the current state of understanding, that the majority of programmatic environments will use some form of API.

It is necessary that MeT considers several environments to understand possible technical limitations that may prohibit the immediate development of the wallet solution.

8.2 Wallet shell

The ability to directly access wallet extensions from the wallet shell is seen as the important value-added factor. It is assumed that such access will follow the usage style that is characteristic for the given device while at the same time providing consistent access to extensions.

Assuming a menu-structured user interface (which is the more popular one), the list of extensions that is available for direct invocation should be clustered together for easy access. This can be achieved e.g. by positioning the wallet shell as one of the top-level menu items.

The wallet does not require the wallet shell access to be controlled or implemented by the native code, even though it may be likely the preferred implementation. This wallet shell may be implemented as an application (e.g. Java MIDlet) as long as the integration with the device's wallet shell satisfies wallet requirements. There is no significant difference for the wallet between the access from programmatic environments and from the wallet shell.

8.3 Remote access

Another method to access the wallet is to use one of protocols that are implemented in the terminal. The modern mobile terminal hosts several protocol clients, e.g. GPRS, SMS, MMS, HTTP etc and each one can be integrated with the wallet.

The exact method to access the wallet from the protocol client should be discussed on the per-protocol basis, as there is usually more than one method to map the wallet extension into the protocol framework. For example, the implementation of the HTTP protocol may view the wallet as the local URI namespace or it may view the wallet as a set of plug-in handlers of MIME data types.

Wallet extensions may have specific requirements regarding remote access, specifically when it comes to access control, as the identification of the remote caller may follow different rules than in the case of local clients.

Most of the protocol clients are currently implemented as native so that the integration with the wallet should happen through the native implementation. There is however the growing tendency to build protocol clients as add-in applications, notably visible for example with browsers. In such case there is no significant difference for the wallet between the access from programmatic environments and through remote access.

9. Creating extensions

There are several ways to create wallet extensions, as they depend mostly on the capability of the terminal to handle different environments. An 'environment' is understood here not only as the traditional programmatic environment (e.g. Java, Symbian), but also as any other type of execution environment that may exist in the terminal (e.g. plug-in cards, remote access to server-executable code).

Following is the short discussion of possibilities to extend the wallet by using different environments.

9.1 Programmatic environments

Programmatic environments include all the environments that allow the application to be downloaded and executed in the terminal. Examples are: Java MIDP environment, Symbian environment or even terminal specific ('native') programming environment if it can accept new extensions.

If the extension is executed within the programmatic environment, the wallet provides necessary inter-connections (including such items as data format conversion etc) while the environment is supposed to provide the necessary registration and invocation interface. This interface will usually take on the form of one or more APIs, depending on the environment, but other solutions are possible. Specifically, the interface with the wallet may re-use one or more of existing mechanisms if they fit the purpose.

MeT believes that the most important environments to address are:
- Java environment (MIDP)
- native environment of the terminal
- Symbian environment

9.2 Smart card

The smart card (or multimedia card) offers the opportunity of the execution environment that is local to the terminal yet physically separated from mobile device environments. In this case the terminal does not provide place to execute the code, but rather the place to register and identify those extensions together with the invocation 'stub' that interfaces between the mobile device and smart cards.

MeT observes the development of smart cards that includes regular ISO7816 cards, JavaCards as well as multimedia cards, including Mc-EX (Mobile commerce extension) ones. MeT Wallet provides an attractive method to access different smart cards and integrate them into the unifying invocation framework.

9.3 Remote extensions

Some extensions may reside not on the terminal but rather on the remote server, and may be available only by specific protocols. Those protocols are implemented in the terminal in a form of protocol clients, e.g. HTTP client, GPRS client, SMS client etc.

In case of remote extensions the wallet provides means to discover and call those extensions as if they were local. The actual access is provided by allowing certain protocol clients to register with the wallet so that the wallet is able to properly route the invocation. The actual registration method is specific to the particular client, but it may be expected that the concept of URI can be used extensively.

Note that it is irrelevant for the wallet what is the exact implementation of the extension or what protocol is exactly used. The wallet does not impose or require any standard with this respect, but rather leaves it to each protocol.

Special care should be taken when it comes to the proper identification of extensions, imperfections in connectivity, increased response time and other problems that may arise from extensions not being local.

MeT believes that even though this alternative looks attractive, it requires further studies and will not be the driving force for the wallet concept. Certainly some application areas (e.g. ones that require end to end security to directly communicate with smart cards) may benefit from such configuration.

9.4 Provisioning

The provisioning process is responsible for delivering extensions into its place in the MeT Wallet and making them available for clients to invoke.

There are several methods to provision the wallet, as each environment (and possibly each vendor or terminal model) may employ a different one. Some methods are standardised while other may be proprietary or vendor-specific. Further, the provisioning may happen at factory/delivery (so that the wallet is already fitted with certain extensions), over the air (where the terminal receives new extensions later into its lifetime) or on fixed media (e.g. multimedia cards).

If the extension is not physically located at one of terminal environments (e.g. it is on the smart card or on the remote server), the provisioning may include mechanisms for the wallet to properly discover and identify such extension so that it can be integrated into the wallet.

The wallet does not mandate certain provisioning method for its extensions but it is leaving it entirely to the specific environments. The wallet assumes however that whatever provisioning method is employed, it will be integrated with the wallet so that extensions can be properly recognised, located and registered with the wallet.

MeT believes that the provisioning is an important issue that may be approached for each environment separately.

10. Security and Identification

The architectural separation of clients and extensions makes it essential to provide proper access control. It is specifically important as extensions may handle sensitive data and, despite all the protection, may be abused. For example, if the extension delivers the service of digital signature, such service may be used by the hostile client to obtain approval against the will of the user.

The wallet is designed with the security protection against threats caused by hostile clients. It is further assumed that each extensions will implement its own proper security policy to protect itself against other threats. The wallet does not address threats associated with physical attacks, attacks with user consent or attacks on the terminal platform.

The wallet builds on the security that can be achieved in modern terminals and provides the foundation for terminals with increased security features. The security of the wallet is built on several concepts that are discussed below.

10.1 Trusted Role of the Router

The security model used by the wallet assumes the central and trusted role of the router, as it is the router that collects and implements security policies. This require all parties (clients, extensions, smart cards etc) to trust the router. Even though such requirement may seem to be too high, it is worth noting that already all applications and all smart cards trust the platform to correctly implement the core software.

For example, Java MIDlet trusts the terminal not only to correctly implement the JVM/KVM, but also to properly set-up MIDP2.0 trust domains and manage permissions. The smart card trusts the terminal to properly convey APDU commands between the given application and the card and to securely collect and deliver PIN from the user.

The trusted role of the router indirectly implies that the router should be implemented in the native code of the terminal rather than being the add-in.

10.2 Security and Environments

As the wallet spans several environments of possibly incompatible security models, the wallet itself does not use the single, mandated method to enforce the security or to authenticate clients and extensions. The wallet relies on each environment's security architecture and re-uses its identification mechanisms. For example, for MIDP2.0 environment, each MIDlet can be identified by the trusted security domain it belongs to. For MIDlets from the same domain, if necessary, identification may use for instance the MIDlet' full name.

The wallet defines the minimum set of functions that must be provided by each environment, specifically the ability of the environment to identify the client or the extension. The wallet allows also non-identified parties to use the wallet, even though their ability to provide and use secure services may be significantly limited.

If the client or the extension is implemented as the remote connection, the wallet assumes that the respective protocol client (stack) is responsible for the security of the connection, including the identification of the remote party. Such information can be used to identify the remote party for the wallet.

10.3 Access control

The wallet provides extensive access control features. Each extension may publish its access control preferences, stating what clients may identify it and use its services. The method by which the given extension provides the wallet with its preferences depends on the environment. For example Java MIDlet may use the manifest file to deliver initial set of preferences and then use certain API to modify them dynamically. The smart card may provide its preferences in the dedicated file.

Such access control preferences are used by the wallet to screen clients so that only allowed clients may use extensions. Further, extensions may use more dynamic access control scheme where an extension is queried each time with the identity of the client so that an extension may respond individually for each situation.

10.4 Mutual Identification

Both parties (clients and extensions) are allowed to know the identity of the other party (including an environment it operates in) before proceeding. The wallet provides such information at identification time before the first interaction.

During the identification the client may enquire about the exact identity of the extension discovered or it may restrict the search to extensions of known identities. For example, MIDlet client signed by X may ask only for extensions also signed by X.

If the extension is using the dynamic access control, it will be provided with the identity of the client during identification. Alternatively the extension will receive the identity of the client with or before the first invocation. At this time the extension may still reject the client.

The exact implementation of mutual identification may depend on the actual environment. In some environments not all identification methods may be available, but the wallet requires that regardless of the environment both parties must have an opportunity to require or learn the identity of the other party before proceeding.

10.5 Several security technologies

Understanding the continuous development of mobile terminals, the wallet does not prescribe a fixed or even a minimum required set of security technologies. The wallet allows several technologies to co-exist while minimising the possible negative impact of low-grade technologies on the overall security of the system by providing means for each party to express its security needs and to assess the counter-party.

The wallet by itself does not assess security of different parties. The wallet collects information of used security technologies from all the environments in an environment-specific way. This information is then made available to the other party. For example, the client may not only learn that the given extension is signed by the particular provider but also that such extension is implemented as MIDlet within the MIDP2.0 environment.

11. User interaction

There are several aspects of the user interaction that are relevant to the wallet. The following is a short discussion of those aspects.

11.1 Wallet shell

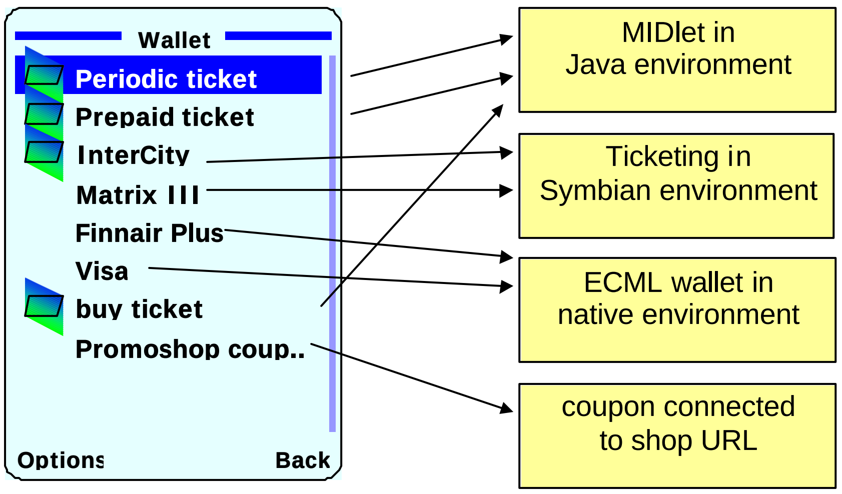
The wallet shell is the component that is visible on the user interface and acts as a client for the wallet, visualising some of the registered extensions. Usually this shell is integrated with the main user interface of the terminal.

The shell can be implemented as a native code or it can be provided as a separate client application, e.g. as Java MIDlet. The wallet shell must be specifically identified to the wallet as a shell, otherwise it will be interpreted as the regular client.

The wallet does not preclude several shells to co-exist. The relationship between different shells is subject to further definition. For example, each shell may define the sub-category of extensions it registers while extensions may provide additional information at registration that may allow the wallet to identify the proper shell to register them with. Alternatively, shells can be specialised to serve the particular customer groups: children, handicapped etc.

Regardless of the opportunity for several shells it is desirable to establish one shell as default one so that an extension is always having certain place to register with. Such default shell may be also used at times where new shells are added or removed. The wallet defines minimum requirements for the default shell. Specifically, the default shell should provide access to all the extension management functions, also for extensions that are not registered with any wallet shell.

Following is an example of the wallet shell that contains multiple registered extensions, implemented in different environments. Some extensions have registered several times with different initial data. The wallet and the shell make use of additional graphical elements to improve the usability of the user interaction.


Wallet
Periodic ticket
Prepaid ticket
InterCity
Matrix III
Finnair Plus
Visa
buy ticket
Promoshop coup..
Options
Back
MIDlet in Java environment
Ticketing in Symbian environment
ECML wallet in native environment
coupon connected to shop URL

11.2 Trusted dialogue

There are several cases where an extension should enquire the user about sensitive information like PIN, pass code and the like. Such interaction requires usually the trusted dialogue, i.e. the set of user interface functions that conforms to certain minimum rules that guarantee that no hostile application is able to copy or intercept the given interaction.

In general, interaction with the user is handled directly by extensions as they are in the position to know exactly when and what data is needed from the user. The trusted dialogue is offered by the wallet in order to augment the existing interaction methods, that are specific to different environments. The use of trusted dialogues promotes the consistency and increases user's trust in applications.

The trusted dialogue is accessible through the wallet's router. As the router, the core of the wallet, is likely implemented as native code, such a solution provides a high level of trust. If extensions have their specific needs to discuss sensitive information with the user, they can communicate with the trusted dialogue through the router.

The format of the user dialogue allows the user to determine whether the dialogue is generated by the router itself, by the native implementation of an extension or by an extension implemented in one of the programmatic environments. In the latter case the user is able to query the interface to determine the origin of the extension which may allow the user to assess the trustworthiness of the dialogue.

Further, the wallet guarantees that no application in the terminal can generate the interaction that is identical with the one created by the trusted dialogue. Therefore the user can be sure about the origin of the interaction. Several different methods can be used to guarantee the ability to distinguish the trusted dialogue.

The wallet also makes sure that implementations of methods used to interact with the user meets all the usual security requirements regarding e.g. keypad feedback, PIN entry, protection against interception/alteration, removal of unused user input, use of predictive input etc. to the extent supported by the environment of the extension.

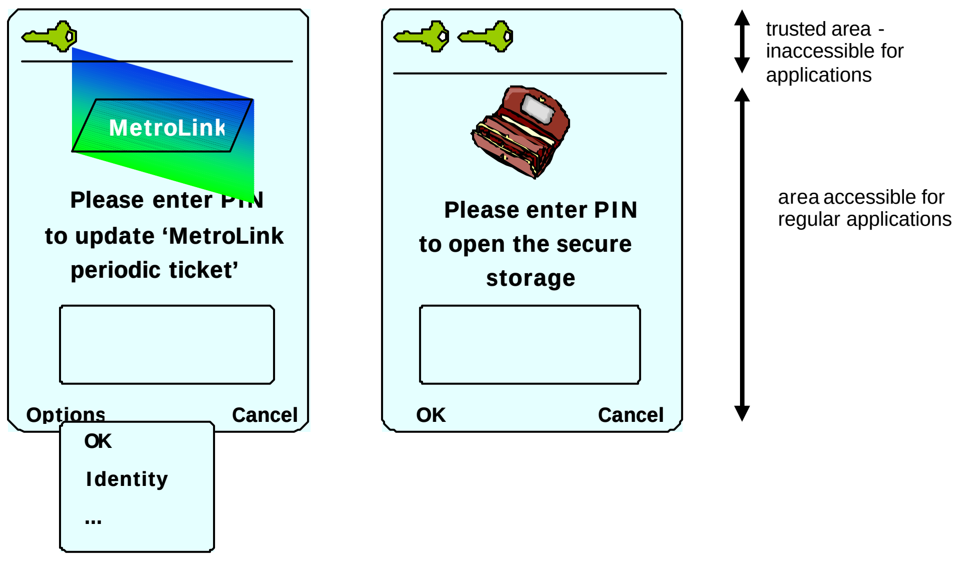
The trusted dialogue is available to extensions by mechanisms that are specific to the environment. The wallet does not mandate how the trusted dialogue must be invoked, but it requires that the originator of the dialogue through such interface must be identified and known to the wallet.

The trusted dialogue can be also called directly by clients. In this case the user will be informed about the origin and identity of the caller if it is known to the wallet (i.e. if it can be reliably extracted from the extension's environment). Note that the wallet may generally consider clients less trustworthy than extensions.

Following picture demonstrates and example of the trusted dialogue used to ask the user for PIN. The dialogue provides certain elements of the design (the key) that cannot be replicated by any other application. On the left side there is a dialogue generated by the third party extension. The dialogue is having the optional graphical branding element provided by the extension. The 'Options' menu item may lead to the complete disclosure of the identity of the extension that has produced this dialogue, as shown below.

On the right side there is a trusted dialogue generated by the native extension (e.g. the standard extension of the secure storage). The branding element is replaced by the wallet graphics and the key is doubled to indicate increased confidence of the wallet in this extension as compared to the downloadable one. Note also that for native originators the 'Options' menu item is replaced with 'OK" as there is no identity to disclose.

The key symbols represent elements on the screen that are associated with trusted dialogue and cannot be replicated by any other application – they are generated by the wallet (the router). Instead of having some area of the screen reserved for such purpose, the wallet may resort to certain colours, symbols or locations in order to distinguish the trusted dialogue from other dialogues. Alternatively, the user may be able to inquire about the identity through the unique combination of keys. All of those solutions satisfy the requirement for the trusted user dialogue.


MetroLink
Please enter PIN
to update 'MetroLink
periodic ticket'
Options
Cancel
OK
Identity
...
Please enter PIN
to open the secure
storage
OK
Cancel
trusted area -
inaccessible for
applications
area accessible for
regular applications

12. Role of smart card

Smart cards are important to the wallet concept. However, due to the complexity of the smart card itself, there are several roles the smart card can play in the wallet at the same time.

The smart card is usually understood as the card that is compatible with the ISO7816 suite of standards. Such card usually contains its own microprocessor and memory housed in the plastic enclosure. Such card may be inserted and then removed from the mobile terminal. For mobile terminals the SIM card is probably the best known smart card.

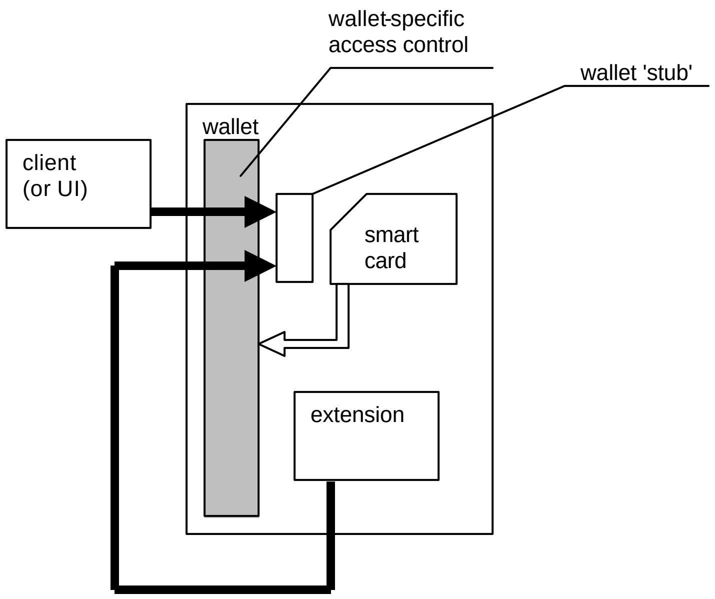
Recent development of smart cards has increased the popularity of JavaCard, the smart card that is able to interpret Java code, becoming the open platform by itself. Further, the success of multimedia has brought plethora of large-memory cards, coming in different formats: MMC, SD, Memory Stick and the like. Such cards can be also fitted with their own microprocessor and essentially provide the functionality equivalent to traditional smart cards.

For the MeT Wallet, both types of cards: smart cards and multimedia cards can be treated in a similar manner. Even though some types of cards may be more suited for particular task, this chapter will deal with both smart cards and multimedia cards under the collective name 'smart cards'.

The wallet sees the smart card as yet another extension that can be accessed through the wallet itself. Such extension is available for both clients and other extensions.

12.1 Smart card as an extension

Smart card may expose its services directly to the wallet so that applications that reside on the smart card can be seen as wallet extensions. This may require the wallet to provide a kind of 'stub' to interrogate the card about available applications and to provide required translation between client call to the wallet and card communication. Further, the wallet may facilitate the access to card from both clients and extensions through the same wallet interface.

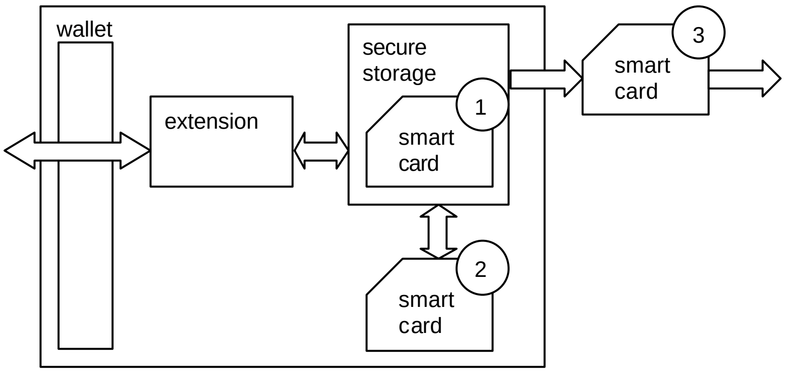


The card application can be called by clients as if it is a regular extension of the wallet. If the card application allows, it can be called directly from the wallet shell, providing the instant integration of smart card and terminal with no additional software element involved. An extension may call the smart card through the wallet as the client does.

This role may be interesting for smart cards that provide standardised applications, as they benefit from very simple integration with other applications in the terminal. Automatic service discovery may be available and the user may be informed about the new functionality immediately once the card is inserted.

12.2 Smart card and secure storage

There may be several different relationships between smart cards and secure storage. Different possibilities are depicted below.



The concept of secure storage does not preclude the usage of smart cards to augment the storage nor does it make it mandatory. The smart card support for secure storage functionality is optional and the usage of it is within the discretion of the extension (the implementation of the secure storage). The caller that is using the secure storage should have sufficient means to decide whether the secure storage it wants to use must or should utilise the smart card support, if available.

There are three basic scenarios of the smart card to support secure storage. First, the smart card can be used as the secure storage itself (or as a part of it). Such storage may provide the ultimate in security, as the smart card is likely to be more tamper-resistant than the terminal. In addition such element of the secure storage is removable, adding the feature of portability.

Second, the smart card can augment the secure storage by providing the needed cryptographic protection. In this scenario the actual data remains in the terminal, encrypted, while the smart card holds necessary cryptographic keys and provides encryption/decryption mechanisms. The secure storage can work properly only when the smart card is present even though data itself is not portable.

Finally, the smart card can be used as a secure container to transfer data from one secure storage to another. Data can be stored into the smart card (and possibly removed from the secure storage) in one secure storage and then unpacked into another one.

12.3 Role of SIM

SIM (recently USIM or UICC when referred to the card platform) is the most popular smart card in mobile communication. From the wallet concept perspective SIM is an important smart card as it is present in all GSM terminals.

The wallet does not assign any special role to SIM, but it may help leverage the multi-application nature of USIM/UICC either by adding extensions that allow clients securely access SIM functionality or by directly integrating selected SIM applications with the wallet for immediate visibility to clients and end user.

13. Secure storage

Secure storage is an extension that implements the collection of storage technologies with improved security over the regular storage provided by the terminal. The extension (or the client) may use such storage to securely keep sensitive information without resorting to solutions that, for instance, utilise the smart card.

The wallet does not mandate or restrict the actual selection of technologies provided that the caller is aware of the actual technology used to secure the storage. In order to promote portability the wallet may recommend certain technologies and defines minimum requirements for such technologies.

It is expected that a popular solution for the secure storage will be the regular terminal storage in encrypted form. PIN provided by the user can be used to encrypt/decrypt data (acting as the main or the only source of randomness). Certain elements of hardware support can be added to this scheme to further increase security. Multimedia cards can be used as another implementation of secure storage.

Most of technologies will likely use PIN to protect access to secure storage. The wallet will support PIN-based protection, but it leaves details of the actual PIN policy (length, change, blocking/unblocking) at the discretion of the given storage. The wallet defines elements of the user interaction (trusted dialogue) that increases the consistency of PIN usage.

The secure storage can be made available to extensions through different means. For example, in the Java MIDP environment secure storage may be implemented as a separate API or as an extension to the existing RMS system. The wallet defines how the secure storage can be accessed within the given environment. At the end of this chapter there is a short discussion of the possibility to re-use the wallet router to access secure storage.

Access to the secure storage is usually controlled by the way the storage is implemented in the given environment. The wallet defines minimum requirements for access control so that data stored by one extension cannot be accessed by another one without proper authorisation.

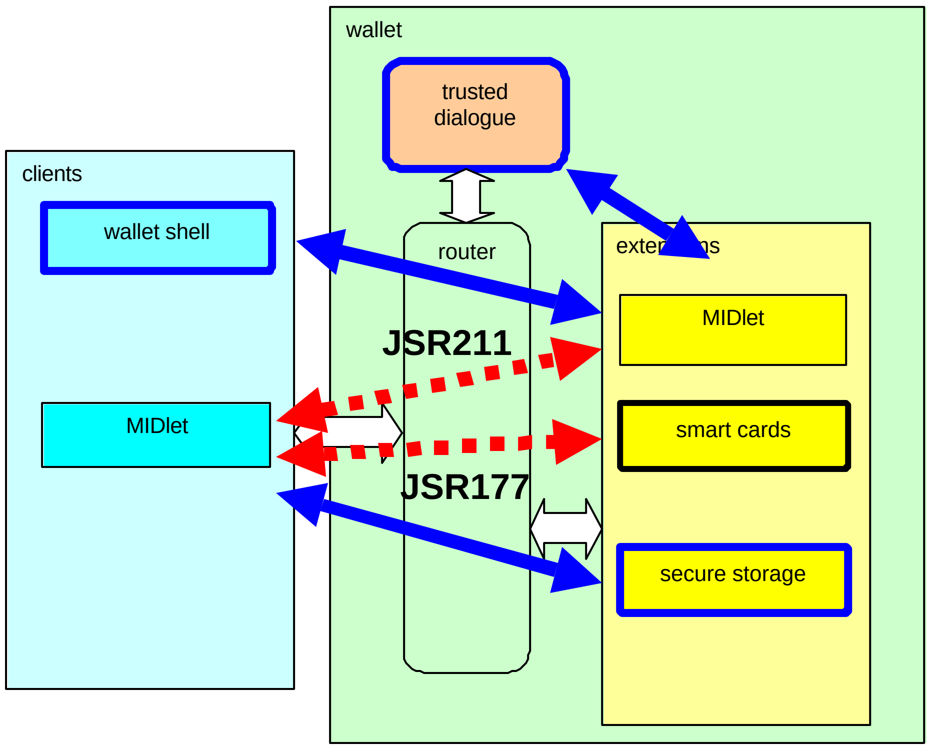
In several environments clients may also be able to take advantage of the secure storage directly, without resorting to extensions. The wallet concept does not preclude such solutions, but it recommends to use the complete architecture of clients and extensions to fully insulate secure storage from possibly hostile applications.

14. Deployment

The concept of the wallet is touching several aspects that are relatively complex to standardise and implement. It also overlaps with several other issues. Therefore it is unlikely to achieve the complete wallet solution in one step. It is therefore proposed to develop the wallet incrementally, through generations. The following is the proposal for the first generation of the wallet.

The first generation of the wallet should be built for environments that are expected to be most popular. It should also utilise existing development to the extent possible. Therefore the following assumptions regarding the first generation are in place.

- *Router*. The router should be implemented in the native code. The router may consist of several independent components (not one monolithic application) that provide wallet-specific functionality.
- *Trusted dialogue*. The trusted dialogue should be addressed mostly by clarifying the implementation of current dialogue methods and built in native code.
- *Configurations*. The ability for the MIDlet client calling the MIDlet extensions depends on the development on Java platform in the mobile device and is seen as an important mid-term goal. Due to current limitation of the mobile device, the most promising configurations for the first generation are:
  - Wallet shell calling MIDlet extensions
  - MIDlet clients calling native extensions
- *Native extensions*. It is seen that two types of native extensions are most important:
  - Access to smart cards
  - Secure storage
- *Wallet shell*. It is assumed that the wallet shell for the first generation will be implemented as the native code. The ability to replace shells is important as the mid-term goal.
- *Access control*. The first generation may provide only certain access control, depending on existing capabilities. For example, the mutual identification may be not available for some configurations of clients and extensions. The complete access control is the important long-term goal.



On the picture above white arrows indicate interfaces that are necessary for such minimum configuration. Red (dashed lines) arrows indicate interfaces that exist or are likely to exist. Each red arrow is labelled with the name of the standardisation group (within Java JCP process) that is relevant to the given interface. Blue (solid lines) arrows indicate interfaces that do not exist. Elements with bold blue (solid) edges require further definitions.

Both components and interfaces are described below. For each item the current status and expected work is determined.

14.1 Router

The router is the core of the wallet offering, providing registration, interface and identification. It should not be assumed that the router must refer to the single module of the native code. Contrary, the router can be implemented in several fragments, depending on current needs.

As the first generation of the wallet concentrates on the MIDP environment, the router may consists of mechanisms that may come from several separate standards within JCP. Some of the interfaces to the wallet may be embedded into the MIDP specification; some may leverage works of other JSRs while yet another may require additional standardisation efforts. It is essential, however, that all those solutions will be coherent with the vision of the router.

The first generation of the router will therefore provide the following set of features, regardless of the number and type of actual interfaces.

- *Registration*. The ability for the MIDlet-based extension to register with the wallet is essential for the wallet to operate. At least static registration should be available, i.e. the MIDlet should be able to register at the time it has been loaded. Dynamic registration is desirable. Note that native extensions naturally may have the convenient access to the registration feature through their implementations.

- *Identification*. The identification may be limited in the first version, i.e. the caller should know in advance the identifier (e.g. the name) of the extension. Such simplified identification can satisfy several use cases.

- *Invocation*. At least the invocation from the wallet shell should be supported. The ability to call extensions from MIDlets may be initially restricted to some extensions only (e.g. secure storage and smart cards), depending on the ability of the MIDlet to call another MIDlet.

- *Access control*. Certain form of access control should be available. In case of MIDlets calling native extensions (smart card access and access to secure storage), access control similar to the proposed one should be present. Mutual identification may not be fully available, but at least an extension may publish its restrictions regarding access. Such restrictions should be observed by the wallet.

The router should be implemented as native code, to provide the highest possible level of security and trust.

14.2 Trusted dialogue

The trusted dialogue provides the ability for extensions to communicate with the user in a secure and trustworthy manner. Several aspects related to such communication have already been presented, whether it is the reassurance that the dialogue cannot be intercepted by other applications or whether the user is properly informed about the origin of the dialogue.

There are some issues that should be clarified.

- *Means of achieving the trustworthiness*. The concept of trusted dialogue requires the dedication of either the designated screen area or some key combination to make sure no application can replicate the dialogue. As both the screen real estate and key combinations are scarce, such requirements may not be easy to satisfy.

- *Preventing interception*. There is an existing body of requirements regarding e.g. secure PIN entry, but innovations characteristic to the mobile device may not be entirely taken into consideration. New interaction modes like predictive input, re-use of numerical keys to enter alphanumeric characters may contribute to the complexity of the problem. Further, mobile devices may contain more than one execution environment with possible interdependencies regarding keyboard input and screen output. Finally, alternative input methods (e.g. voice, joystick) should be considered.

- *The consistency of the user experience*. It is widely accepted that the user should face a similar dialogue when handling a similar task. Such consistency can be achieved within the single device across all the environments, but it can be also achieved within the same environment across different devices.

14.3 Trusted dialogue for MIDlets

MIDlets currently are having a wide selection of methods to interact with the user, including several text-only dialogues as well as colour graphics, animation and similar. Specifically, there is a selection of input modes to address sensitive inputs like passwords or PINs.

The current MIDP2.0 specification allows for several different implementations. Some of valid implementations may not be secure enough or trustworthy enough for the wallet. On the other hand, the existing set of methods is already widely known by the developer's community.

It is therefore recommended for MeT to evaluate the most appropriate implementation within the existing MIDP2.0 specification and adopt such implementation for the wallet. This is the task similar to one that has been undertaken earlier regarding the consistency of the user experience.

14.4 Wallet Shell

The wallet shell is responsible for presenting to the user extensions that has registered with the wallet and allowed for user activation.

The architecture of the wallet shell is under discussion. Specifically, the following items are under consideration.

- *Initial shell*. The initial shell is the wallet shell that is active immediately upon the initial power-up of the new device - the built-in feature of the device. The existence of the shell is not strictly required to make the wallet operational, yet it is of significant benefit. The user is actually aware of the existence of the wallet only if there is a certain element of the device user interface that can be clearly associated with the wallet. Further, some extensions may immediately expect the shell to exist so that they can register to the shell and be invoked by the user.

  It should be considered whether the device should be shipped with an initial shell (the shell provided by the manufacturer), whether such shell can be loaded at first power-up (possibly provided by the distributor) or whether the shell may be loaded later at the user's convenience (so that it can be delivered by third parties).

  It is suggested that the device will be shipped with the initial shell, provided by the manufacturer. Such solution allows all the extensions to assume the known initial configuration of the wallet regardless of the user's choice. Further, such shell may offer an access to wallet management functions, thus creating the single access point for the user to all the wallet features.

- *Updating the shell*. Once the shell is present in the device it may be possible to update the shell e.g. to offer the lifestyle choice or additional features. The ability to update the shell provides the user with the value of personalisation, while at the same time offering different players the ability to deliver specialised (e.g. branded) shells.

  As the shell gets updated some issues regarding the relationship between old and new shells should be considered. First, it should be decided whether the new shell replaces the old one or whether two or more shells may co-exist together. If there is more than one shell present at the same time, the relationship between them should be clarified - whether there is only one active shell at any time and whether the user can select another shell as active. Finally, it should be decided whether there is the default shell, the non-removable shell that resides (semi) permanently in the terminal.

It is recommended that the device will allow for the default shell (that is the initial one) always present at the device. In addition, the device may allow for more shells to be present at the device but only one of those should be active (i.e. handling the interaction with the user). This architecture solution allows for the flexibility (as shells can be changed) while protecting the user interaction (as the default shell is always present, even though it is not always active). Mandating only one shell to handle all the interaction greatly simplifies the design and provides the user with the consistent experience with the shell. Specialised (branded) shells can be still provided by dedicated clients.

- *Shell and clients*. The shell is one of clients that can use the wallet, yet it requires special handling by the wallet. Differences between the shell and the regular client should be clarified. The most important difference is in the dynamics of the registration process, as the shell should be able to accommodate all the changes to the registration as they happen while the regular client can only identify the current set of extensions. Further, the extension should be able to determine its visibility to the wallet shell regardless of the identity of the shell.

  It is recommended that the wallet shell should preserve its special status so that not every client can act as a shell.

The following architectural solution provides the summary of proposals from the discussion above. Further discussion is necessary to reach the common understanding.

The default shell, preferably implemented in the native code will be present at start-up as the initial shell. The default shell will not be removable but the user may install another shell that will replace the default one. If the user-installed shell is removed, the default one becomes an active shell again. If the user installs another shell and makes it active, the previous one becomes inactive so that at any time there is only one shell that is active.

14.5 MIDlet shell registration

The ability to call the MIDlet extension from the shell is one of the main configurations for the first generation of the wallet. This includes the MIDlet ability to register with the wallet and then the ability of the shell to invoke the MIDlet.

Currently MIDlets are usually started from some form of the shell that is available through the regular user interface of the device (e.g. through the menu). MIDlets that are loaded to the device are therefore registered by default with certain shell. This solution may not be sufficient for the wallet. The following issues should be addressed.

- *Position of the wallet shell.* The wallet shell should be well presented to the user on the device's Its position, visibility and accessibility should be at least as good as for the shell that is currently used to launch MIDlets. Further, the relationship between the MIDlet shell and the wallet shell should be clarified.

- *Static registration*. It is seen that the static registration (register at load time) is essential for the first generation while the dynamic registration (change registrations while running) is the mid-term goal.

- *Additional information*. In order to facilitate the usability of the wallet shell it is advisable to provide certain graphical elements that can identify the extension, in addition to usual textual names.

- *Structured registration*. It may be of value to allow for structured registration, i.e. to create sub-menu structure of the shell so that the extension may not only register to the wallet, but also register to the particular slot within the wallet.

The static registration may utilise information that is embedded together with the MIDlet code in the jar file, as one or more of MIDlet attributes. Existing attributes (like MIDlet-Name or MIDlet-Icon) can be re-used immediately for the purpose of the wallet shell while other features of the shell (registration itself, structured registration) may require standardisation of additional attributes.

Currently there is no identified activity that may lead to the development of the MIDlet mechanism that may allow the registration of MIDlet to the wallet shell, whether statically or dynamically. There is certain longer-term potential in the JSR211activity, for shells that are implemented as MIDlets. MeT will work with JSR-211 Expert Group to initiate the activity.

14.6 MIDlet accessing other MIDlets

Limitations of the majority of MIDP implementations, specifically in terms of device's resources, currently prevent MIDlets from running at the same time, thus rendering MIDlet to MIDlet communication very hard to achieve. At the same time, however, this communication is beneficial for several possible application areas. For example, the browser-plug-in (MIME content handler) can be implemented by MIDlet that can then be called from the browser implemented as another MIDlet. Further, in the absence of arbitrary class loading, this mechanism may be beneficial for the deployment of component-based applications.

The ability to call MIDlet extensions from MIDlet clients is the important mid-term goal. Currently there is an activity within JCP, JSR211 that may eventually lead to the creation of the respective access.

14.7 MIDlet access to Smart Cards

Smart cards (in several formats) are playing an important role in security and are ideally sited to augment the wallet solution. It is seen as important to allow MIDlets to access smart cards that are present in the mobile device.

The existing initiative within JCP, JSR177, provides the complete solution for smart card access (in addition to its other components). Specifically, communication methods for ISO7816 cards and JavaCards are provided together with access control. MeT will explore whether the solution provided by JSR177 is sufficient for other types of cards (MMC, SD etc).

14.8 Secure Storage

The secure storage can be implemented in software alone (or in software with hardware support) and it can be used to store confidential information. Known solutions use the PIN (or password) as the main source of randomness as they encrypt confidential information with the key that is derived from such PIN/password. Even though the software-only secure storage cannot provide tamper-resistance comparable with smart cards, it is an attractive option for solutions where security requirements are moderate or where cheap solutions are sought.

The concept of the secure storage is quite flexible so that it brings several issues that should be discussed.

- *Algorithms*. Even though cryptographic algorithms have become pretty standard, there are a lot of variants and variations that may impact the quality of the implementation. Also the way algorithms are combined into the complete solution may have a significant impact on the security as well as on the usability.

- *Role of the hardware*. The security of the storage can be greatly increased by using some hardware elements, whether it is the dedicated hardware storage, key storage or crypto-processor.

- *Device dependence*. The secure storage may benefit from device dependence (e.g. it may use the unique device identity to augment the PIN), but at the same time it may hamper its ability to transfer data between devices in encrypted format.

- *PIN/password rules*. As the storage builds most of its security on the PIN or password, strict rules should govern the selection of PIN. Specifically, the PIN policy: minimum PIN length, character base, PIN blocking, multiplicity of PINs etc are up for discussion together with PIN implementation issues: PIN storage, PIN verification etc.

- *Communicating security level*. Different devices may possibly provide implementations of the secure storage that may significantly differ in security. The developer should be able to decide whether the security level offered by a particular implementation is sufficient (assuming the trustworthiness of the implementation itself). This can be done e.g. by requesting the minimum security level or by allowing the client to identify the security level that is actually offered..

Certain simple versions of secure storage are commercially available on mobile devices, either as built-in or as add-on. Some of the standards also address software-only storage for confidential information. MeT will explore methods used in order to develop the recommendation for secure storage.

14.9 MIDlet access to Secure Storage

The MIDlet specification already defines the storage that is accessible for MIDlets, the RMS (Record Management System). Such persistent storage uses the device memory to hold data that is (in the simplest model) private to the MIDlet that created it.

The implementation of secure storage can be done either through a special API or by differently leveraging the existing RMS solution. All options are worth considering.

- *Special API*. The special API may define the storage that is independent from the RMS, thus freed from all the legacy issues. This allows to address properly all the issues related to the secure storage provided by the wallet, including mutual identification, resolvability etc.

- *RMS integration*. The concept of extending RMS builds on the understanding of RMS by the developer community. By adding certain security options to RMS it may be possible to achieve the same quality of the solution without fragmenting the area of storage solutions, but at the expense of waiting for another release of MIDP.

- *MeT-specific RMS implementation*. As the RMS specification does not define any security measures, it is possible to define the secure specification of RMS within MeT. By reserving certain part of storage namespace and certain properties the API may remain unchanged.

Currently there is no identified activity that may lead to the development of the MIDlet access to secure storage. It is therefore suggested that MeT will take action to initiate such activity.