Developing a Transaction Engine for Mobile Payment

Andrei TOMA, Radu CONSTANTINESCU, Floarea NĂSTASE
Academy of Economic Studies, Bucharest, Romania
andrei.toma@ie.ase.ro, radu.constantinescu@ie.ase.ro, nastasef@ase.ro

With the ubiquitous presence of mobile phones, there is increasing economic viability of a framework for service and product payment through mobile devices. Such a framework involves on one hand software running on the device itself, but more importantly a back end with sufficient flexibility to cover the wide range of possible transactions. The back end is materialized through a generalized transaction engine which operates on abstract operations. Constructing the engine involves identifying the possible actors and operations which the system must support as well as formalizing a possible architecture for the engine itself. The engine must support complex multi stage operations involving the external actors.
Keywords: *Mobile Transactions, Operation Engine, Flexible Architecture*

1 Introduction

The goal of the system is the construction of an intermediary payment layer for mobile phone transactions. Transactions are initiated by software running on the mobile phones themselves or on specific devices (such as electronic points of service).
In order to preserve system flexibility the implementation must be done through a general enough mechanism that will allow it to be extensible. The reason for this approach is that the technologies involved in mobile payment are extremely dynamic and may lead to different interactions within the lifespan of the platform.
Communication with the system must be flexible enough to support requests from a multitude of devices. Also, some actors, such as financial institutions, will always be outside the system. For these reasons, the interface with the system is done through a series of web services.

2 A mobile payment system

In order to implement an abstract mobile transaction layer, a series of prerequisites must be taken into account. First of all, given the diversity of the involved actors, the mechanisms selected for implementation must allow for a uniform standard, independent of the particular implemented use case.
The second important aspect is that the system must be capable of running a diverse set of operations and that the full set of the necessary use cases is not known from the beginning, a fact which suggests a solution with a high degree of flexibility as pertaining to the possible operations and the activities that compose them.

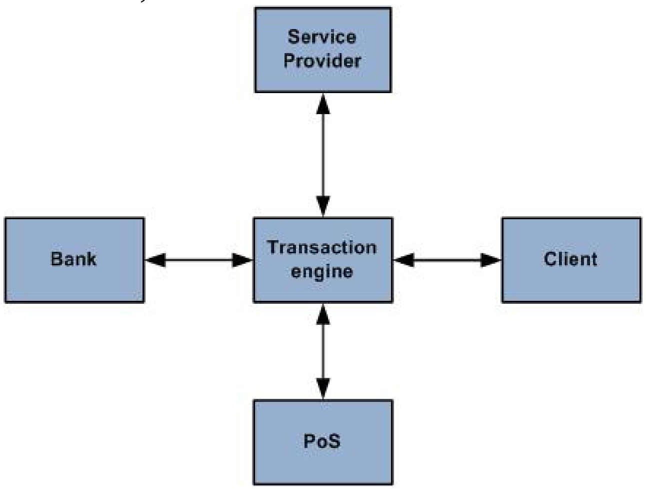

Fig. 1. Actors in the system

3 Identifying the actors

The first step in implementing the system is identifying the relevant actors which are involved in a series of operations which involve a varied degree of complexity. Some of these actors are registered with the system while some are external.

The typical client of the system is an individual who wants to make rapid transactions through a mobile device. While this type of operation is at the core of the system, the supported operations are defined in a manner allowing initiation through a varied set of means (e.g. the portal associated with the system) without significant changes. The clients register to the system through the portal.

A Point of Service (POS) is a special type of client to the system as although the POS is an intermediary for the client's transaction, it can be assimilated with an ordinary client, albeit registered through administrative means and not directly. The significant differences will refer only to the rules pertaining to authentication in the system.

A service provider is a legal person registered with the system to which payments can be made by the clients. The service providers also have direct access to the system and are registered through the portal.

Banks are financial institutions to which the system can send letters of payment in the name of the clients. Banks are not registered with the system and the particulars of the involved communication mechanisms depend on each bank.

4 Common use cases

While the scope of possible operations is at first glance very large, the substantial differences between them are generally restricted to the communication channel through which they are initiated. For example, an operation representing payment through SMS, initiated by sending a message to a special number is, from the point of view of the operations engine, identical with payment through the portal. Although the initial request is different, it can be converted into a message sent to the system through a web service. For the above reason the two operations would be identical, assuming they refer to the same operation.

Considering this initial analysis, the main operations covered by the engine are payment from a virtual wallet present in the system, accompanied by the possibility of placing funds in the virtual wallet, and payment from an external source of funds (e.g. a bank account).

Below we detail a series of use cases which are implemented in the system. While these use cases are fairly specific, they are actually representative of the main classes of operations which the system can cover.

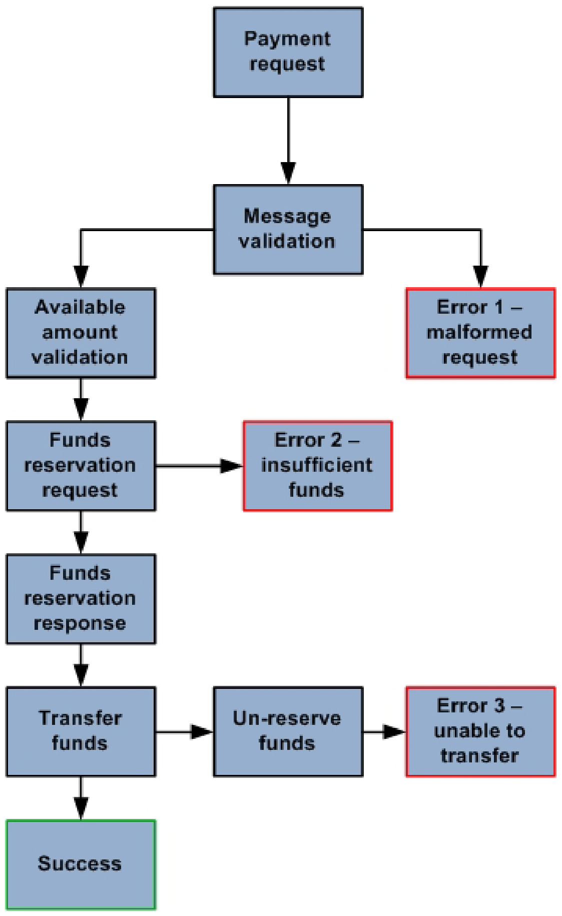
Payment request
Message validation
Available amount validation
Error 1 – malformed request
Funds reservation request
Error 2 – insufficient funds
Funds reservation response
Transfer funds
Un-reserve funds
Error 3 – unable to transfer
Success

Fig. 2. The flow of an SMS payment

Figure 2 presents one of the main use cases reflected by the system. While the figure refers to payment of a service via SMS, from the point of view of the transaction engine, this particular payment is equivalent to any payment from a virtual wallet. The activities with a red border are final activities corresponding to errors, while the activity with a green border, is a final activity corresponding to the successful completion of the operation (which must be unique).

The use case is triggered when the system receives a payment request, if the payment is to be performed with funds from the virtual wallet. After the system receives it, the request is validated with two possible results. If the request does not contain all the necessary fields, an error is returned, signaling the user that the request was malformed.

If the request was valid, the system goes to the next processing step, consisting in validating the requested amount against the available amount in the virtual wallet. Again, if the funds are insufficient, an error is returned signaling the user that the wallet did not contain the required amount. If the validation is passed, a request is done with the system to reserve the funds for payment.

The next step is the payment itself which can either be performed without problems, leading to a successful final activity or lead to an error signaling the user that the payment could not be performed. In this event, the reservation on the funds is rescinded.

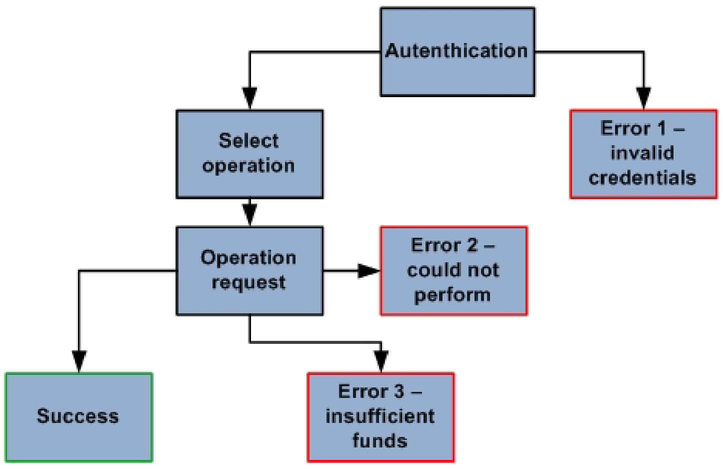
Autenthication
Select operation
Error 1 – invalid credentials
Operation request
Error 2 – could not perform
Success
Error 3 – insufficient funds
Fig. 3. The flow of a portal operation

Figure 3 presents the flow of an operation performed through the support portal. Although the use case describes the acquisition of a service through the portal, from the point of view of the engine, this operation is equivalent with the payment through any mechanism.
The use case is triggered when the user accesses the portal. A preliminary step consists in the authentication of the user, which can either have a negative result, in which case the system returns an error message, or positive, in which case the system continues to the execution of the requested operation.
The operation can have multiple steps and depending on its results, multiple errors can occur. In this particular use case, the possible errors were an error reflecting the absence of sufficient funds and a more general error in the execution of the operation.
If no errors were encountered, the use case ends in a success, otherwise an error message will be returned to the client, reflecting why the operation could not be completed. The possible causes depend on the nature of the operation itself.
In the case of the acquisition of a service, for example, the system might not be able to perform the payment due to lack of funds. Since the execution of the operation is done through the support portal, some possible errors specific to the operation might not occur. For example, since the format of the request is controlled, it is improbable that the request is malformed. However, since these checks are done in the operation itself, the operation is still capable of returning the corresponding error message.
Figure 4 presents another important use case implemented in the system, payment of a service from funds present in a bank account. Although the use case describes the acquisition of a service through Bluetooth, from the point of view of the transaction engine, this operation is equivalent to any payment requested through any mechanism, as long as the necessary funds are in a bank account.
The use case is triggered when the system receives a payment request. The activities present in the figure are typically preceded by the authentication of the client.
The activity in the operation is the validation of the message form. If the operation has not received all the information necessary to perform the payment an error is returned to the client, signaling that the request was malformed.
If the request was valid, the next step is to validate the amount of available funds. In this case, the validation involves communication with the bank. A request is made to the bank to reserve the necessary funds. If the response from the bank reflects the absence of sufficient funds, an error message is returned to the user.

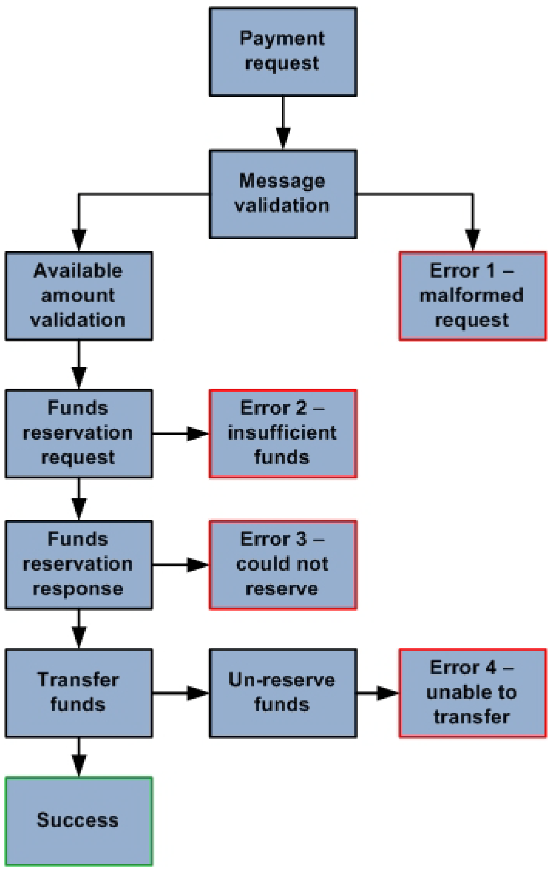
Payment request
Message validation
Available amount validation
Error 1 – malformed request
Funds reservation request
Error 2 – insufficient funds
Funds reservation response
Error 3 – could not reserve
Transfer funds
Un-reserve funds
Error 4 – unable to transfer
Success

Fig. 4. Flow of a bank account payment

If the amount has been successfully reserved, a request to transfer them is made to the bank. Depending on the response from the bank, this will either end the operation with a success, or continue it. If the transfer was impossible, it is still necessary to instruct the bank that the funds are no longer needed, after which an error message is sent to the user, informing it of the impossibility of the payment

5 The operations engine

5.1 Concepts

At the core of the system is the operations engine which is responsible with the execution of specific operations requested from outside the system. Execution is done in a predefined order of activities by navigating their structure based on the feedback received from actors outside the system.

An operation, and thus a set of activities, is started when a request is received from outside the system through the web service which is exposed to the clients for this purpose.

The operations engine executes abstract operations, composed of abstract activities. The particular activities composing an operation are executed in the order of the activity flow defined within; execution of atomic activities stops when a final activity is encountered.

From the engine's point of view, the content of each activity in the flow is unknown as only their outcome and the order it determines is relevant.

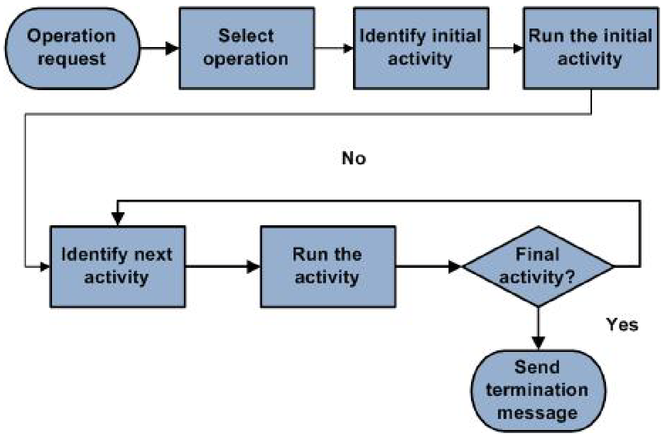

Fig. 5. Flow of an operation

The following sections take a closer look at the principles by which the engine functions and the concepts that they are based on.

5.2 Abstract operations

An abstract operation is the representation in the system of an atomic exchange of data referring to a payment, a transfer of funds etc. in a manner ensuring a high level of generality. The advantage of this approach is given by the possibility of future implementation of a large variety of operations without modifying the engine's implementation.

The model of the interactions between the system's clients and the other involved actors is assumed to be in constant evolution and thus an operation must be defined in such a way as to be able to abstract any future interaction. An operation is thus an entity which can be run by the engine regardless of its content.

Operations are represented as entities which receive a message and initiate the execution of an activity flow starting from an initial activity. The operation maintains an activity tree and is responsible with identifying what activity follows a completed one.

When an operation is started, it identifies the initial activity from the activity tree and sends it the parameters necessary for its execution. After a particular activity has started running the operation waits for the outcome of its execution. Depending on the result the operation receives, the operation selects the next activity and runs it. The execution of the operation stops when a final activity is encountered.

5.3 Abstract activities

The representation of an activity within the system should ideally be abstract in the sense in which any previously implemented activities can be assembled into an operation. An advantage of this approach is the high reusability associated with an activity.

From the view that activities are just exchanges of messages with actors from outside the system the advantages are limited. However, abstract activities can have diverse types and serve diverse purposed. For example activities can be defined which handle data conversion or filtering.

Special purpose activities include initial and final activities. Initial activities are the entry points of operations and an operation has only one initial activity. Initial activities are identified by the fact that they have no parents in the activity tree.

An operation stops its execution when it encounters a final activity which is an activity with no children in the activity tree. Almost invariably, a final activity will consist of communicating the outcome of the operation to one of the involved actors.

5.4 Logging

An operation also handles logging its own start and all the activities that it executes in a transaction log. This has on one hand the purpose of ensuring that should the system not perform as predicted any errors can be corrected and, on the other hand, creating the possibility of transaction reversal. This would be desirable when, although there was no error in the functioning of the system, the

external reason for the operation made it invalid (e.g. factual error, fraud).
In the latter case, it would be desirable that operations in the system were reversible. This can be achieved by ensuring the activities themselves can be executed independent from one another (atomic).
If activities are atomic and the order of their execution is known, the system can reverse an operation.

6 The support portal

The engine is coupled with a support portal which represents the interface through which both the clients and the service providers can register, make payments and view their transaction log.
Although the main purpose of the portal is registration with the system, due to the way in which the operations engine communicates with the exterior, the functionality of the portal can be extended to also include all the supported operations.
The portal covers user authentication, registration of clients and service providers (coupled with the validation of the supplied registration data) and launching operation requests to the engine. In a similar approach to the operations engine itself, all portal functionality is accessible through web services.

7 Engine technologies

In the implementation of the engine the technological choices were dictated by the general approach. As such, both the engine and the portal communicate with the exterior and each other via SOAP 1.2 web services. This ensures that the engine can be used by a variety of possible client applications. The services themselves are written in the Java language with the JAX-WS framework. In order to expose the services, a GlassFish application server was used.
Data access is handled through JDBC, ensuring a minimal of data model abstraction; in the prototype phase, the data is stored on a MySql database server.
The portal is constructed in JSF(Java Server Faces) over a service model (all portal operations are also done through web services) also deployed on a GlassFish application server.

8 Conclusions

The prototype of the transaction engine implemented in research project […] has a series of advantages. From the point of view of the inherent dynamic of transactions involving mobile devices, the engine was implemented in and extensible manner in order to easily cover new operations as they become necessary. More importantly, the implementation of new operations would only involve adding a new series of activities. Even that can be avoided if the necessary activities have already been implemented as part of previous operations.
Activities are a flexible mechanism through which diverse transformations of the data can be reflected as well as validations etc. allowing for the construction of complex operations. Furthermore, the representation of the activities in an actual operation has the added advantages of being simple and flexible.
Communication with the system is also done in a flexible manner, with all the requests in XML format. This approach ensures that the transaction engine can be invoked from any type of client application even if it is built using radically different technologies from the ones used in the implementation of the system.

Acknowledgements
This work was supported by PN II research contract nr. 3039/01.10.2008. We would also like to take this opportunity to thank our students, Ioan Dragan and Alex Stănimir, for their important role in the implementation of the prototype.

References

[1] SERAFIMO, *Integrated platform for electronic financial transactions and services implemented using current mobile devices technologies*, PN II research contract nr. 3039/01.10.2008, project director Prof. Dr. Năstase F., stage III "*Industrial research – implementation of the experimental model*"

[2] SERAFIMO, *Integrated platform for electronic financial transactions and services implemented using current mobile devices technologies*, PN II research contract nr. 3039/01.10.2008, project director Prof. Dr. Năstase F. stage II "*Identification of the technologies used in the development of the project*"

[3] SERAFIMO, *Integrated platform for electronic financial transactions and services implemented using current mobile devices technologies* PN II research contract nr. 3039/01.10.2008, project director Prof. Dr. Năstase F., stage 1 "*Industrial research– analysis and design*

Andrei TOMA graduated the Faculty of Cybernetics, Statistics and Economic Informatics, Economic Informatics specialization, within Academy of Economic Studies Bucharest in 2005 and the Faculty of Law within the University of Bucharest in 2005.In present conducting doctoral research at the Academy of Economic Studies.He is currently interested in IT Law as well as Computer Science issues and seeks an interdisciplinary approach to legal issues related to Computer Science.

Radu CONSTANTINESCU has a background in computer science. He has graduated the Faculty of Cybernetics of the Academy of Economic Studies in Bucharest. He has finished his doctoral research at the Academy of Economic Studies, with the topic on Information Security. His fields of interest include computer security related issues.

Floarea NASTASE is a professor at the Economic Informatics Department at the Faculty of Cybernetics, Statistics and Economic Informatics from the Academy of Economic Studies of Bucharest. Competence areas: information technology and communications, computer net-works, web technologies, e-business, information systems audit. She is author or coauthor of 16 books and more than 80 scientific papers published in national and international conferences proceedings and journals. She participated in more than 15 research projects, as director or as team member.