Developing a Micropayments System in Local Area Networks

Cristian GEORGESCU
cristian.georgescu@ugal.ro
Dunarea de Jos University of Galati, Rodica

Abstract. Starting from the solution of aggregating micropayments in a local area network that operates with reduced costs, with simple and easy to use equipments, with security procedures that are not so complicated, based on the rapport between the cost of fraud and the expected value obtained through fraud, a first stage of developing this system has been accomplished and materialized in the analysis stage by using some of the diagrams put to use by UML.

Key-Words: Electronic payments, Micropayments, Electronic money, Digital wallet, Local Area Network, Traceability, Informatics system, UML.

JEL Code: L81, L86

1. Introduction

A transaction involving small amounts of cash, although advantaged by the fact that cash has no cost or is the bearer of small withdrawal commissions and also advantaged by the possibility of dividing and aggregating until reaching a level considered acceptable by all partners, are not free from inconveniences. Cash has a friendly behavior towards fiscal evasion and its use is not traceable.

This proposed system [6] simulates an electronic micropayments system by having one, equivalent in its purpose, which manages all sales within a local area network and makes all payments at the end of the period. Within traditional commerce, we find this situation at the “neighborhood groceries” where local customers, known as loyal customers may credit a number of small transactions with the payment at the end of the month. In this case, the LAN is an information system at the neighborhood level and the authenticity and solvability is determined by interpersonal relations, whereas the security and confirmation of all transactions are replaced by mutual trust. This solution treats the way several “neighborhood groceries” operate: serving not only their customers, they can introduce certain commercial crediting facilities and can have a quantitative and value tracking of their activity.

This system, represented in Figure 1 through a use-case diagram [3], presumes the existence of an administrator, as trust figure, that ensures merchants and customers registration in a local area network, with a server, a communication system and simple equipment, for the merchant in order to introduce the sum and customer’s confirmation. This software will identify the transaction, both parties, will aggregate all data and manage the customer’s guaranty for final documentation.

Figure 1. Use-case diagram for proposed solution [6]
It is my appreciation that such a system represents one possible solution for the private business environment involved in managing chains of neighborhood stores, malls, markets, etc. but also for local public administrations that can track small retail commercial activities that take place in public spaces under its administration.

2. Partner registration and transactions.

As said before, all sellers and buyers must be registered into the system, like all parties that know each other in the “neighborhood groceries” model. The difference between the traditional system and the one presented consists in the information depositing space, identification and solvability of all partners. In the traditional system holding this information is distributed between the partners: each vendor knows their customers and each customer knows the merchants with which they interact. In the electronically support system we opted for the solution of centralized storing of information regarding the partners with network access possibility for identification and validation. For the distributed version, it is imperative to have more powerful and safer remote equipment.

Developing this solution has been done by detailing the scenarios with state diagrams. In Figure 2, the use-case “Register” from Figure 1.

The use-case “Transaction” describes commercial interaction between the vendor and buyer. It is in a dependence relation with the use-case “Settlement” showing that any modification from the transaction implies a corresponding adaptation of the settlement between partners. The dependence between “Register” and “Settlement” use-cases functions in the same manner. The detailed presentation of the transaction was made with the state diagram from Figure 3.
As prefigured in [6] the system is distributed using mobile equipment for the merchants, having the role of clients that allow communication with the server on which the main processes take place. The proposed solution is that of design by component.

3. Component oriented software

Development, maintenance and adaptation of a complex and big system apart from being a difficult process, it is also a big time killer. Thus, developers have constantly looked for methodologies that would help them reduce the size of projects by reusing existing code or by acquiring certain modules that can easily be integrated into their applications. Additionally, this methodology should permit extension of reusing this code. [9]

Splitting the project into logical components is the essence of analysis and object oriented development. This also constitutes the basis for software components that is made of reusable bits of binary code that can be integrated in other components from the same system or from a different one, with a relatively reduced effort. [8]

Achieving the component oriented software is based on two programming paradigms: encapsulation and dynamic-link. The first defines object oriented programming while the second has been a permanent concern of programmers while perfecting development techniques for the software.

It is important to know that an approach based on software development components does not dictate the structure of the application, but rather the model is the one making it possible to program and use certain components of binary software that can be in an independent evolution. These components are independent from the application that uses them, and also from programming languages used to create them.

The main objective of component oriented programming is the interoperability that, simply put, means the ability of different objects to work together. Objects are predetermined, their source code is not accessible or necessary and the programming language that uses them may be different from the one that is used to make them. For reusing binary code it is then necessary to define an interface through which the object can be accessed.

For ensuring encapsulation, the obvious solution consists of separating the interface specificities from the implementation details. The interface must only define the methods that clients may call upon,
the internal processing logic being inaccessible to the caller. In this way, perfecting internal processing is ensured without affecting the caller.


System security has to be strictly guaranteed because the sum of all transferred values can become a considerable one, but the balance between security procedures' complexity and the importance of the elements protected by these procedures must also be taken into account. The security component, implemented on the server, has two interfaces for connecting to the validation procedures specific for customers and vendors. It is presented in Figure 4.

```
Figure 4. Component diagram for security subsystem
```

System's security is ensured by the collaboration between the mobile equipment with its sub-functions and the security component from the server. According to acceptable costs for the hardware option, the security solution can be implemented from a simple version, with a password assigned to each buyer, to a more complex version with authentication using fingerprints reader. The simple version, with a password for each buyer can be improved with a sequence of passwords that the buyer uses successively.

```
Figure 5. Component diagram of payment aggregation
```

The aggregation component actually makes the concept of efficient micropayments possible. All payments made in the local area network, by a buyer towards any vendors are processed after validation, stored and, at the end of the shopping from the micropayments system, are aggregated and a single payment is made, totaling the whole amount to be paid. For the vendors, a different income aggregation is made representing the volume of activity and, finally, the corresponding sums from the total payments made by each buyer will be distributed.

```
Figure 6. Component diagram of quantitative management
```

The quantitative management component is an optional one because it can track, in parallel with each micropayment, the measuring unit for the product acquired. Data that identify vendors and buyers, already existent, can ensure delivery of a rapport, on demand, with the quantitative and value situation for vendors or articles bought.

5. Conclusions

As we can see from the deployment diagram from figure 7, the system uses the client-server architecture. The server contains components for security, aggregation and management and the customer may use these components through remote hailing mechanisms.

The following implementation details will determine whether to create an out-of-process server - an executable component that can be on the remote computer, but also on the local machine or an in-
process component – a dynamic linked library. Communication between clients and these mentioned components can be achieved through the “Automation” facility. [8]

The “Automation” facility allows applications to expose their functionality in order to be used by other programs. Each object supports specific properties, methods and events, allowing client programs to use all these features. An application that is normally run by the end-user will be able, in this way, to be controlled by another program, without further involvement of the user.

The application can show its abilities through the user interface. In this way, each component will define its own set of user interfaces. For many applications this is not a problem, should each client program be designed to work with a certain set of interfaces. Still enough programs have to administer a wide variety of components, from which many may implement user interfaces that don’t even exist when the programs are written. In these cases, it is difficult for a client application to access a user interface without having any prior information. The solution may consist in using libraries called “type library”, that are binary files which store information of the properties and methods of the object in a form accessible for other applications, at run-time. Using type-library, an application can determine what interfaces can an object support and in what way it can hail their methods.

Such a configuration will allow use of the system starting from a small number of users, to a very large one, without prior information or preparation. In other words this ensures the scalability of the system.

Anticipating malfunction situations triggered by communication errors or incomplete data transfer would make the existence of roll-back procedures for databases on the server in case the clients would fail in totally transferring an entire data lot.

Instantiating server components only when they are hailed by clients facilitates “just-in-time activation”, combined with the reduced set of transferred data for each transaction, they will together raise the system’s security.

The technical solutions presented will have to be reevaluated after appreciating the investment effort and having a cost-benefit analysis starting from market studies and official statistical data.

References
[17] ***, www.mfinante.ro - Proiect de lege privind activitatea de emitere de monedă electronică
[18] ***, M. Of. nr. 446 din 1 iulie 2010, Ordonanta de urgență a Guvernului nr. 61/2010 pentru modificarea Ordonantei de urgența a Guvernului nr. 113/2009 privind serviciile de plată