A Cluster-based Payment Gateway System Developed Using a Distributed Functional Language

Carlos Abalde, Víctor M. Gulías, José L. Freire, Juan J. Sánchez

LFCIA, Computer Science Department
University of A Coruña, SPAIN
{carlos, gulias, freire, juanjo}@lfcia.org

Abstract

In this paper, the design and implementation of a payment gateway system that acts as a mediator among an e-Commerce, a final customer computer and a bank host is presented. The gateway, designed using modern software engineering tools (unified modeling language, design patterns, etc.) and implemented using a functional language (ERLANG/OTP), supports the typical features of such systems but pays a lot of attention to its architecture, using distributed computing to improve service availability (scalability, fault-tolerance). The architecture and ideas proposed in this paper are intended to be a first step towards the implementation of other and more complex payment systems.

1. Introduction

In this paper, the design and implementation of an electronic payment system over Internet, based on an innovative architecture that can be reused for other similar systems, are described. The proposed system can be divided into three main components:

1. The payment gateway system, which is the main focus of this work, is a distributed application implemented using a functional language (ERLANG/OTP) and deployed in a cluster of computers.

2. The point of sale (POS) system, an interaction facade library for the payment gateway system. It has been implemented in several languages (PERL, C++ DLL and JAVA) in order to make easier the integration with the existent e-commerce products.

3. The host adapter, used by the payment gateway for the access to the financial networks. The adapter acts as an abstract layer that hides the concrete details of each financial entity host.

The payment gateway is the main and more complex system. Its key features are:

• Execution in a cluster of computers, using a schema describing how the system load is going to be distributed among the cluster nodes.

• Scalable. A new set of nodes can be added to the cluster without changing the system implementation. The system is able to satisfy the requirements in a simple context (downwards scalability); and is able to increase the performance by adding new resources to the architecture, serving more concurrent users (upwards scalability).

• Fault tolerant, meaning that if one of the nodes of the system architecture crashes, the users are not going to notice any problem with the service. Whenever this happens, another node is going to assume the role of the crashed in a transparent way. With 24x7 expected uptime, there is a need for some kind of mechanism to keep working at least in a degraded mode when some kind of fail happens.

• All the internal state of the system is stored into a distributed database whose configuration and distribution ensures that the distribution of the database is not going to be a limitation for the gateway scalability.

• The system is able to interact with different kind of devices, such as Web browsers, WAP terminals, or any other that could appear in the future, without changing the internal implementation.

• Personalization of the contents that are shown to the clients depending on the needs of each virtual shop contracting the services of the payment gateway.

• Low cost. The goal is to satisfy all the above mentioned requirements with an architecture that can reduce the implementation and deployment costs.

The paper is structured as follows. In section 2, the state of the art and motivations for the proposed system are described. In section 3, the key technologies used during the
design and development of the system are discussed. Then, a first approach to the architecture of the payment gateway is described in section 4. Section 5 explains more in detail some of the refinements and improvements of the system. Finally, section 6 shows some performance measurement results.

2. State of the Art

Traditional means of payment suffer from various well-known security problems: money can be counterfeited, signatures can be forged and checks can bounce. Electronic means of payment (electronic payment systems) retain the same drawbacks. In addition, they entail some additional risks: unlike paper, digital “documents” can be copied perfectly and arbitrarily often; digital signatures can be produced by anybody who knows the secret cryptographic signing key; a customer’s name can be associated with every payment, eliminating the anonymity of cash, etc [1]. Nevertheless, a well designed electronic payment system can be as secure as the traditional ones, or even more, as well as being more flexible.

Since mid-nineties, dozens of electronic payment systems which use public insecure networks as Internet had been proposed. Only some of them have been implemented. A classification of the more relevant systems and projects could be done using the following categories [1, 2, 3, 4]:

- **Support for the use of card numbers in public networks**: ad hoc solutions based on SSL, Net Market, CyberCash, SET...
- **Electronic intermediates**: VirtualPIN, NetBill, VirtualCash, PayBox, Yahoo! PayDirect, PayPal...
- **Electronic checks**: FSTC Electronic Check Project, NetCheque...
- **Electronic cash**: CAFE, Mondex, NetCash, CyberCoin...
- **Micropayments**: Millicent, SubScrip, MicroMint...

The payment gateway proposed in this paper falls in the first category, but with an innovative architecture comparing with other solutions with similar functionality. The architecture is intended to be a first step towards the implementation of other payment systems.

3. Key technologies

Besides the innovative architecture presented, three of the most important and differentiating features of the proposed solution are the use of a distributed functional language for the implementation, the adaptation to a LINUS cluster-based architecture and the intensive use of design patterns.

3.1. Erlang/OTP

The programming language used for the implementation of the payment gateway, ERLANG/OTP [5, 6], has been designed and used in Ericsson’s computer science lab (CSLab) for developing distributed control systems. The combination of the functional paradigm and the parallel computation defines a declarative language, with almost no side effects and with a high level of expressiveness, abstraction and ease for prototyping.

ERLANG/OTP is specially suited for soft real time, distributed and fault tolerant systems. The language is based on asynchronous message passing, transparent communication of values, high order communications, and it is designed to support a high amount of concurrent processes. The language is suited for distributed systems development, and allows transparent allocation of processes among different nodes. Besides, ERLANG/OTP includes primitive functions for supporting fault tolerance and allows code replacement without having to stop the system.

3.2. Linux Cluster

The use of Beowulf clusters (low cost LINUX-based architecture) as part of the proposed solution is one of the keys that makes the approach innovative. The main advantages of this technology are the existence of an OpenSource community with experience in high speed networks, distributed systems and clustering, the availability of the source code and development tools, the ease for migrating to other UNIX versions, and the support for several platforms.

3.3. Design patterns

The design patterns are a concise and elegant way for representing architectural concepts that are commonly used by experienced software designers. They give the designers a common reference framework and a common terminology, being useful for improving the understanding and for reusing the knowledge acquired in past successful developments.

The concept has been borrowed from the civil architecture [7] and has been a revolution in the way of understanding the software design, specially after the collection published by a group of authors known as the gang of four (GoF) [8].

Typical ERLANG/OTP applications are designed as a big number of small concurrent processes. Despite of the simplicity of individual processes, the global behavior of the ERLANG/OTP system can be very complex. This idea suggests a decomposition guided by the use of design patterns, from the adaptation to a distributed platform of the classical GoF patterns to other low-level language-specific patterns,
available in this case as behaviors (generic servers, supervision tree, generic finite state machine, etc.).

4. Design and architecture overview

The payment gateway went through several refinement steps, from the initial prototype, satisfying most of the requirements identified during the analysis, to a final system with the desired features of distribution, scalability and fault tolerance, all with a reasonable low cost architecture.

4.1. Interaction among the agents

Starting with the basic interaction among the three main agents in the payment system (POS, payment gateway and customer - figure 1), going through a phase of requirements capture, a sequence, XML [9] structure and content of the exchanged messages using UML sequence diagrams. Afterwards, state diagrams were used for formalizing the state changes of an electronic transaction, both in the POS and the payment gateway.

The outcome was a communication protocol among the agents, where different cryptographic techniques (symmetric and asymmetric cryptography, digital envelopes, etc.) and XML messages digitally signed, encapsulated and transported using HTTP secure connections (SSL) were used.

XML was chosen for representation of the messages to simplify the interoperability of the agents, and because it is a newer, more flexible and clearer alternative to ASN.1, used in systems like SET [10]. All the messages follow the structure depicted in figure 2, and include the digital signature of the sender, calculated by using a simplification of the DOMHASH algorithm [11].

4.2. Separation between content and view

With the goals of making the gateway able to integrate the design of the contents showed to the customers with the design of the virtual shop; and allowing the use of different access devices (WEB, WAP, etc.), the gateway was designed for generating intermediate results as XML content, that is translated to a concrete format (HTML, WML, etc.) using XSL style-sheets [12].

4.3. Host adapter

The access of the payment gateway to the financial network is done by means of the host. With the goal of abstracting away from the concrete details of the host used by each financial entity, the access interface is proposed as a strategy pattern [8], implemented as an interchangeable module that is part of the generic server state.

4.4. Initial prototype

Putting away the POS, because it is a simpler software component that acts as facade for the payment gateway, the combination of all the ideas introduced in this section gave place to the initial prototype of the gateway. In this initial version, still not distributed and with reduced fault tolerance features, the same technologies that in the final version (ERLANG/OTP, MNESIA distributed database, XML/ERL XML parser, the XSL processor SABLotron, the HTTP server INETS and the cryptographic library OpenSSL) were already used, and all the requirements identified during the analysis phase were already covered.

5. Design refinement

The next step was to evolve the initial prototype of the payment gateway to a flexible, scalable and fault tolerant system.

5.1. Decomposition into subsystems

The initial gateway, with a monolithic architecture, evolved towards a new version decomposed into three large subsystems:

- Virtual shops access subsystem, composed by a pool of nodes, each of them executing a lightweight HTTP server and with specific access controls (access lists, SSL authentication of the clients, etc.) for replying POS’s queries.
• Customers access subsystem, composed by a pool of nodes, each of them executing a HTTP server which answers the customers requests. Both this subsystem and the previous one can be composed by more than one node. The load balancing schema is out of the scope of the payment gateway, being a simple reasonable option to use the round robin DNS giving a unique point of access from the outside.

• Transaction processing subsystem, composed by a pool of nodes in charge of processing the transactions. This subsystem has been decomposed into a variable number of subsystems, in order to avoid that the use of the distributed database could cause scalability restrictions in the gateway (section 5.3). Each of these subsystems is composed by a variable number of nodes, balancing the load among them following a master-slave architecture, where each slave decomposes the processing of each request into the steps shown in figure 3.

5.3. Scalability

The final prototype can scale in several dimensions, depending on the design needs:

• The virtual shop access and customer access subsystems have unlimited scalability.

• New transaction processing subsystems can be added with no limitation to the system.

• Each transaction processing subsystem can scale independently, with the unique restriction of the restrictions related to the distributed database. The writing actions in a distributed database with a high number of nodes can be too slow, due to the need of locking all the other nodes with writing access to the involved tables. The transaction processing subsystem writes very often to the database in order to maintain the transaction state updated, and this could produce performance limitations. In that case, instead of adding new nodes to the existent subsystems, it would be a better idea to add a new processing subsystem with the new nodes, thus doing a partition in the distributed database.

6. Performance

In order to check some of the features of the system, some benchmarks have been done in LFClia’s cluster, Borg (figure 4). This cluster is composed by 23 internal nodes and a front end, all of them connected by a double Fast Ethernet 100 Mbps network (channel bonding) with four 24 port switches. The switches themselves are connected by a 1 Gbps network.

Figure 5 shows how the incorporation of new nodes to the processing subsystem originates a linear reduction of the response time and, therefore, an increase of the number of requests processed by second. Additionally, the figure shows that there is a minimal processing time by message that cannot be reduced adding nodes to the subsystem. This time only can be reduced optimizing the processing steps (parsing, translation, optimizing cryptographic operations, etc.).
7. Conclusions and future work

The payment gateway implementation covered the expectations of its design. The use of ERLANG/OTP as the implementation language was a good decision to build a scalable and distributed system, with an innovative architecture that can be reused to develop more complex payment systems.

The ERLANG/OTP alternative could have been PVM or MPI, which allow the implementation of distributed systems, but to make them scalables it is not an easy task. Another option could be the use of a distributed object technology such as CORBA or RMI. With a good design we can build a distributed and scalable system, but its implementation would be more complex for sure.

Additionally, choosing of ERLANG/OTP as the implementation language gives to the payment gateway support to uninterrupted operation and fault tolerance, in a simple way. Moreover, another advantage of using ERLANG/OTP as the underlying platform for the system development, is that techniques coming from the formal methods area in order to improve the system design and performance [13], and to ensure (at least partially) the system correctness [14] can be applied. These techniques can be used more naturally when taking as input a high level declarative language as ERLANG, and their results are specially appreciated in security critical applications as a payment gateway.

Finally, the combination of XML and XSL stylesheets to the communication between systems has showed to be the best option to simplify its interoperability and to support tasks such as the content personalization and the independence of the access protocol.

Independently of the improvements over the specific payment system, the architecture could be extended and improved in aspects such as the load distribution among the cluster nodes, the improvement of the performance in some parts of the system, the support of hardware cryptographic devices, the refinement of the personalization features or the the application of any XML digital signature standard [15].

8. References


