A Case Study on Legacy Information Systems Migration: A Fault Tolerant Approach

Rafael de Paula Herrera[1], **Alan Salvany Felinto**[1]

[1]Department of Computing – State University of Londrina (UEL)
Londrina, PR – Brazil

herrera.rp@gmail.com, alan@uel.br

Abstract. *Legacy Information Systems play key-roles on organizations development and growth. However, they can be considered as risky factor to operations chain whether they do not meet the demanding or become acting as single point of failures. In this work, we propose a migration model which is able to handle systems that depend on Relational Databases and its changes were driven through the use of a distributed middleware. We also pose how this approach was successfully applied while migrating a Legacy Information System to a Cloud Computing based infra-structure, adding fault-tolerance to its architecture as a competitive advantage, enabling the related services to be clustered and then horizontal scaled on demand. All major concerns on how the whole solution and its aggregated tools were conceived are discussed in high-level details, so them can be solely reproduced and integrated to another systems in order to achieve the same goals or improve its level of quality assurance.*

Resumo. *Sistemas de Informações Legados são peças-chave no desenvolvimento e expansão das organizações. No entanto, podem ser considerados fatores de risco para a cadeia operacional se não atenderem a demanda ou se agirem como pontos singulares de falhas. Neste trabalho, propomos um modelo de migrações capaz de lidar com sistemas que dependem de Bancos de Dados Relacionais, tendo suas mudanças sido conduzidas por meio da utilização de um middleware distribuído. Também mostramos como esta abordagem foi aplicada com sucesso durante a migração de um Sistema de Informações Legado para uma infra-estrutura baseada em Computação nas Nuvens, adicionando tolerância à falhas como vantagem competitiva, possibilitando que os serviços relacionados fossem clusterizados e então escalados horizontalmente sob demanda. As principais preocupações sobre como toda a solução e suas ferramentas agregadas foram concebidas são discutidas com um detalhamento em alto-nível, de modo que possam ser individualmente reproduzidas e integradas em outros sistemas, visando atingir os mesmos objetivos ou melhorar os níveis de garantia da qualidade.*

1. Introduction

Information Systems play key-roles on sustainable evolution to practically every industrial segment. Their continuous use represents Return of Investment (ROI) as one of the most tactile results along productive chain [Vemuri 2008].

Over the time, most of Information Systems (IS) tend to be classified as Legacy Information Systems (LIS) due to several reasons, from obsolete Hardware and/or Operating Systems in which they are running on, to social factors as lack of fully understanding about its entire lifecycle or codebase, caused by documentation absence or loss [Brodie and Stonebraker 1995, Bisbal et al. 1999].

In this work we propose a migration model that was developed and successfully applied over production environment in a real time vehicle fleet monitoring LIS. Its application stack ran in a hardware that was highly susceptible to failures and yet could be smoothly moved to a Cloud Computing infrastructure without services disruption.

A conceived distributed middleware [Herrera and Felinto 2012] heavily influenced the design of migration model presented here. We also pose its base architecture details along qualitative results using a set of auxiliary components that was developed in order to assist the systems integration with its Service Provider (SP).

In section 2, we present all related works that influenced us during the architecture designing, which also increased our vision about the development of auxiliary solutions that could allow a secure evolution of the present proposal, while applied to ASP in production.

We introduced the overall environment in section 3, reporting in details the main interdependence relationship found on services provider. Basically, blocks portrays how different systems integrates themselves into a single high-level solution, which end-to-end is taken in details in order to provide us a wider comprehension about the criticality on changing the whole chain.

During the analysis of all systems which composes the solution, we rose up several questions about critical points that could lead the application to fragility and failures. In order to mitigate such behaviors in an improved infrastructure, there were reported the singular points of failure we found, in section 4, being properly cleared.

With an identified opportunity of improvement and knowing the wish on moving the solution from an obsolete infrastructure to another one which could be based on Cloud Computing, we exposed in section 5 the details of developed and applied migration model.

We emphasize critical questions that motivated the designing of strategies and auxiliary tools, so the process could be concluded without outages on services providing.

Finally, we approach on section 6 our ideas about future works, which we believe to be the next potential contributions to distributed systems architecture area, that once well understood in details, could be applied to innumerous LIS from several kinds.

2. Related Works

Maintaining a LIS requires concerns on data interchangeability between related systems, so several approaches were described by [Bisbal et al. 1999]. Its main contribution to our research was the properly discretization and analysis of migration strategies, exposing qualitative up/downsides from each one of them. Inspired in a set of techniques and best practices stated in, we developed a distributed middleware which main focus is to take care of performance issues on LISes that strictly depends on relational databases, providing a codebase foundation to its evolution [Herrera and Felinto 2012].

Assuming that information systems evolution is directly related on how hard is its decomposability [Brodie and Stonebraker 1995], a good migration strategy should be focused on turning a big and monolithic solution into smaller pieces of software, which could be easily managed, tested and atomically changed over the time, however the qualitative features that determine such level of success on this approach are not explicit and needs a specific study on the complexity carried by legacy solution [Sneed 1995].

In order to lead a data migration, there was developed a methodology concerned on it and the need for parallel operation of the legacy and target systems during migration was discussed in [Wu et al. 1997]. We advocate that a gradual migration model that should retains both systems running until the whole process is done, as a secure and reasonable choice. System evolution is covered as re-engineering and continuously improvements by a stream of incremental changes [Warren and Ransom 2002].

There was developed a middleware framework that links a new Web-based user interface with a wrapped Legacy Information System, as an incremental migration strategy based on re-engineering of user interface [De Lucia et al. 2008].

Reusing existing concepts of old methods allowed us to re-engineering them in new methods while some server regions was adapted for WAN data interchanging through distributed middleware adoption [Brinkkemper 1996, Herrera and Felinto 2012]. In a similar approach, a resource access middleware was developed considering a 3-layer architecture to wrap legacy scientific application to grid service [Lu et al. 2005].

There was presented a survey of key approaches to integrate and/or transform legacy applications into services to participate in an enterprise-wide SOA [Erradi et al. 2006]. Also, a two phases approach integrates legacy systems into Web Services [Parsa and Ghods 2008] was developed nearby the period of a case study on modernizing large scale systems to SOA was made publicly [Vemuri 2008].

A descriptive case study from the same industry segment as our work was fits in, was presented and covers aspects on re-engineering and evolution when Service Oriented Architecture (SOA) is adopted [Nasr et al. 2010]. Migrating application features to SOA by black-box approach based on multi-agent system was presented [Fayçal et al. 2010] and we developed auxiliary tools which essentially has the same influences.

A management decision model was presented for migration versus integration applied to SOA, taking care of combined strategic factors and cost-benefit analysis for key-decisions [Umar and Zordan 2009]. Architecture reconstruction to support system modernization through identification and reuse of legacy components as services in a SOA is shown along an industry case study [O'Brien et al. 2005].

Our testing workbench evolved in a sense that affected server was handled as a blackbox, in order to ensure cohesion over architectural codebase changes. A similar approach was used in [Zhang et al. 2008] while migrating another application class. Testing on SOA is discussed by means of complexity they could eventually impose, as its location is no longer tiered to a single organization domain [Lewis and Smith 2007, Canfora et al. 2008].

3. Overall Environment

The Service Provider domain was composed by dedicated hardware under a single physical cell and hosted heterogeneous systems communicating themselves by means of open and proprietary protocols.

Overall environment is show by Figure 1, where mobile devices send real time telemetry and geolocation data collected by sensors, from A endpoint. Customer Application Servers were located in apparted infrastructures, performing monitoring and control jobs over the vehicle fleet, so we must consider B as a multi-endpoint.

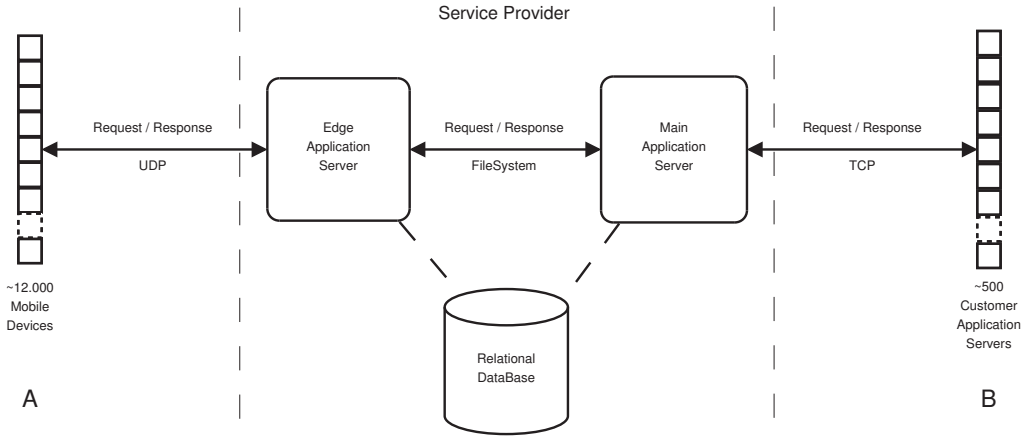

Figure 1. Main structures and its joint points.

There was an in-house message exchanging system between $\approx 12,000$ devices and Edge Application Server (EAS). At least one message per device is delivered within a 90 seconds interval. Using Global System for Mobile Communications (GSM) or General Packet Radio Service (GPRS) means over Transport Control Protocol (TCP) has lead to eventual communication breakdowns, motivating the development of a protocol featuring checksum control and ordered delivery, over User Datagram Protocol (UDP). This strategy was an effective and lightweight approach when compared to previous one, with a low rate of packet loss and communication failures (both bellow $\approx 0.05\%$).

The relationship between the EAS and a single device is shown by the Figure 2, where each device has an internal bus, able to communicates with sensors and actuators which is connected to. EAS retains a complete in-memory states mapping of sensors and actuators and if it does not get changed, in about 10 minutes, it should be persisted into Relational Database (RDB).

Based on the nature of interchanged messages between Mobile Devices, Service Provider and Customer Application Servers (CAS), integration data is generated in plain text files, which would be consumed later and, if needed, key-states of entire solution could be modified into EAS reserved memory and then persisted to RDB.

The Main Application Server (MAS), consumes the available information through a Shared File System (SFS), as shown in Figure 3, sending a bunch of pre-processed files to CAS, which would be used later on building operational reports to its end customers. All the communications between MAS and about 500 CAS was given by means of a plain text protocol, authenticated, cryptographed and TCP based.

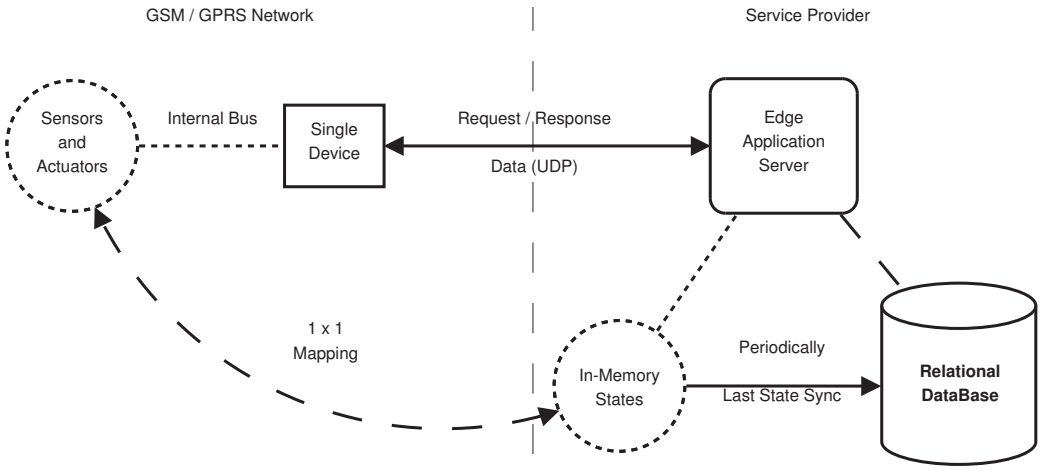

Figure 2. Relationship between the Service Provider and a Single Device.

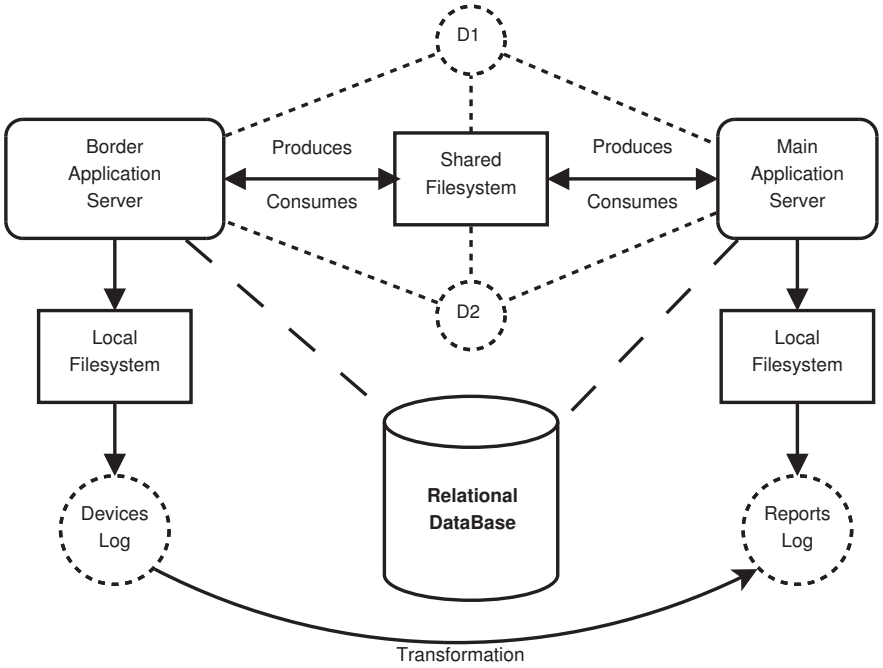

Figure 3. Relationship beteween EAS and MAS, both located at the Services Provider infrastructure.

In order to get integrated to whole solution, each customer should maintain an Application Server under its own infrastructure. It was charged on receiving and aggregating all vehicle fleet data, which was previously designated as its responsibility. The Figure 4 shows how this approach could lead us to an distributed data storage of pre-processed informations, by distinct business centers, once the raw informations are stored on Services Provider side, waiting for data recovery needs that could eventually be requested by some customers, which are limited up to 3 years period.

We can classify all the communications as bi-directional, in other words, the end customer can interact with its vehicle fleet, intermediated by these mechanisms, sending commands that goes from requesting specific data and then reconfiguring specific devices,

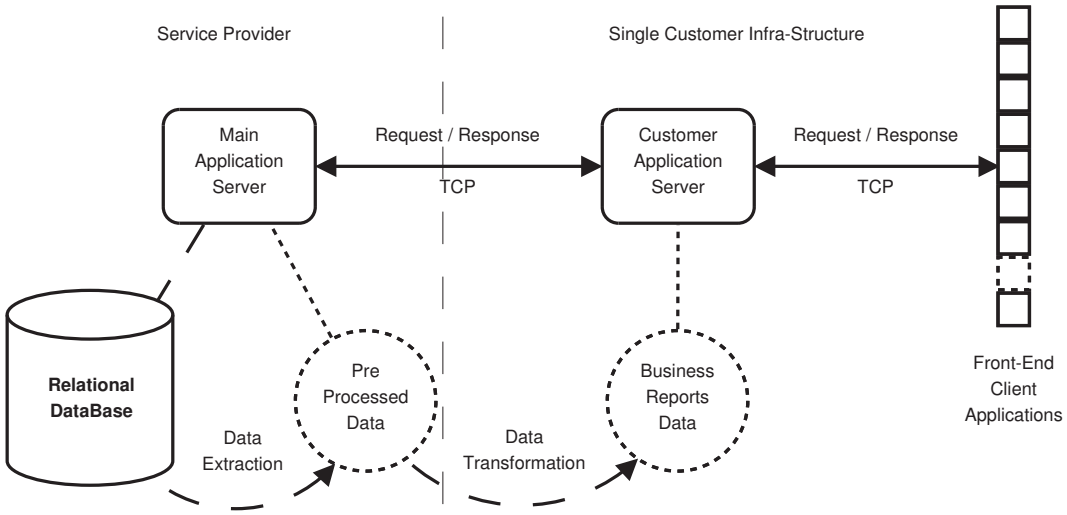

Figure 4. Relationship between the Services Provider and a Single Customer Infra-Structure.

to start actuators which are physically attached to the monitored vehicles. The SFS and the RDB could be changed during this process, and therefore, the associated data flow is variable according to the actions taken by its end users and received data from devices that would be later being processed by BAS.

4. Singular Point of Failure

The main characteristics that lead us to develop a distributed middleware in the past [Herrera and Felinto 2012], was the presence of a data-structure which is essential to EAS working: an operations buffer that should be taken into RDB. Once the service, by any reason, was abruptly broken, such buffer was lost and, consequently, the related operations to missed SQL queries could cause data inconsistencies.

There were threads made for receiving and dealing with arrived data from both ends of BAS, either from mobile devices or customers. We have found 2 failures at the non-transactional employed flow which, together with the stateful nature of crucial data being temporary stored in memory, could cause inconsistencies in durable-state of whole solution in case outages being occurring within key-steps. The flow can be described by the following main states:

1. The first state, Idle, means a waiting period before any checking could be done over EAS received data. It can be configured and by default is set up to 100 ms.
2. Having the Idle period passed by, a Polling is done in order to catch new arrived messages, despite where did they came from. If there are no messages received, the buffer is empty and then the thread returns to Idle state.
3. If a message was received, it will be processed and then got its results written into RDB. Whether the Acknowledge (ACK) is unnecessary, then the messaging process is finished and the thread returns to Idle state.
4. If a received message needs ACK after its results were written into RDB, then a message is sent with a list of actions made over some request and the messaging process is finished, so the thread returns to Idle state.

5. In case of failures within the system fragility period, an inconsistency will be generated in the whole durable-state. This occurrence demands maintenances and could be sporadically found, ever with outages due abruptly crashes.

It is possible to abstract the described flow of messages handling, as shown by Figure 5. Basically, when an end point does not receives the right acknowledge of some written data, a singular point of failure is reached. So on, a set of discrepancies might be found on virtual data mapping, which corresponds to real world states at several levels of the whole solution, depending solely on the criticality of the lost data and what system modules such consistent data affects.

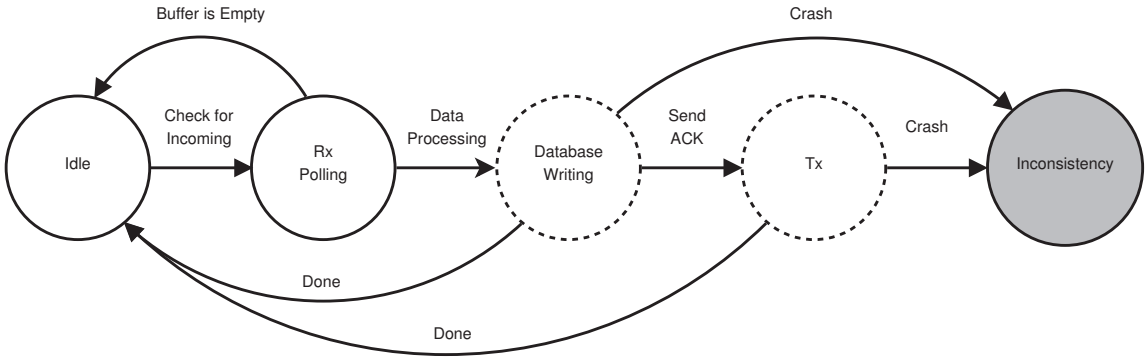

Figure 5. States Machine illustrating an inconsistency upon EAS outages.

Suppose that devices reconfiguration were received by EAS, it would be forwarded through the network to devices, after a properly handling. Once the targets have stated that they properly received and applied the configurations, their related statuses should be persisted to RDB. If, from this time, an outage occurs, the non-transactional nature of the process at both ends, would certainly lead us to an inconsistency.

5. Developed Migration Model

Aiming uninterrupted service providing, we conceived a migration strategy able to turn the Service Provider current architecture into a fault tolerant one. In order to overcome problems such as Hardware provisioning and solution continuous evolving, we used an environment that is solely based on Cloud Computing technologies, understood as Infrastructure as a Service (IaaS) and provided by Amazon Web Services™(AWS).

In order to accomplish such goal with cohesion and security, a smooth migration was planned over the monitored devices. Consequently, the end users base were also gradually migrated. The chosen devices groups, for each migration step, were taken according to complexity level of operations. Each numbered migration step is related to Mobile Devices features and is ordered from less to most critical ones:

1. Only provides geolocation queries.
2. Receives remote reconfigurations on its periodic reports.
3. Retains behavioral changes on its actuators, by remotes reconfigurations.
4. Generates real time critical alerts such as vehicles violations or requested helps.

The Figure 6 relates the percentage of devices and customers that were affected in each migration step. All devices and end users base were migrated from an environment

which was susceptible to failures and ran over obsolete Hardware, to another one that is fully elastic and fault tolerant. However, the fault tolerance itself, could not be achieved only with the full stack of applications being moved to such environment, but it was needed that this Software could be deployed to multiple instances and being able to finish some operations that were running into another machine which suffered from outage.

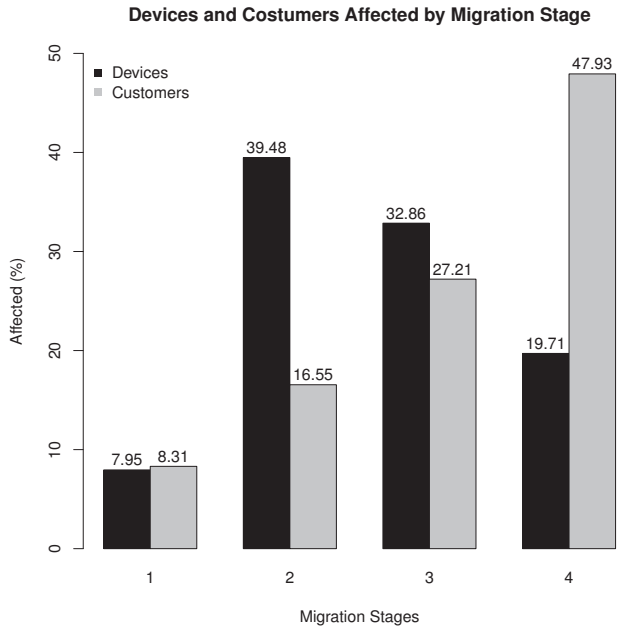

Figure 6. Impact of each migration step on devices and customers.

In order to build the idealized migration model and reach the fault tolerance, it was developed a middleware (Herrera and Felinto 2012) that was able to intermediate the communication process with RDB, storing all the stateful data from EAS into an in-memory grid. It enabled that multiple instances of the same daemon being able to continue running operations from another instances that for any reason were abruptly interrupted.

An overall solution overview, counting on proposed improvements, is portrayed by the Figure 7, where the distributed architecture is shown, while still be able to interchange data with legacy infrastructure. The main idea consists on allowing all the migration steps could occur such way that the services providing would not be shutdown.

The Cloud infrastructure received a modified version of BAS, where it is possible to do a linkage between multiple instances of the same kind, so eventual crashes would not lead to unavailability. While the migrations were occurring, the responsibility over devices was transferred from the 1st RDB to 2nd RDB, once all devices should be reporting all informations with Cloud infrastructure as its main reference.

During the migrations steps, there were several updates made on EAS codebase. To ensure that no feature gets broken, there were developed two simulators, able to reproduce the behavior of end customers and mobile devices in everyday operations. The Figure 8 shows the simulated environment working, where EAS is handled as a blackbox

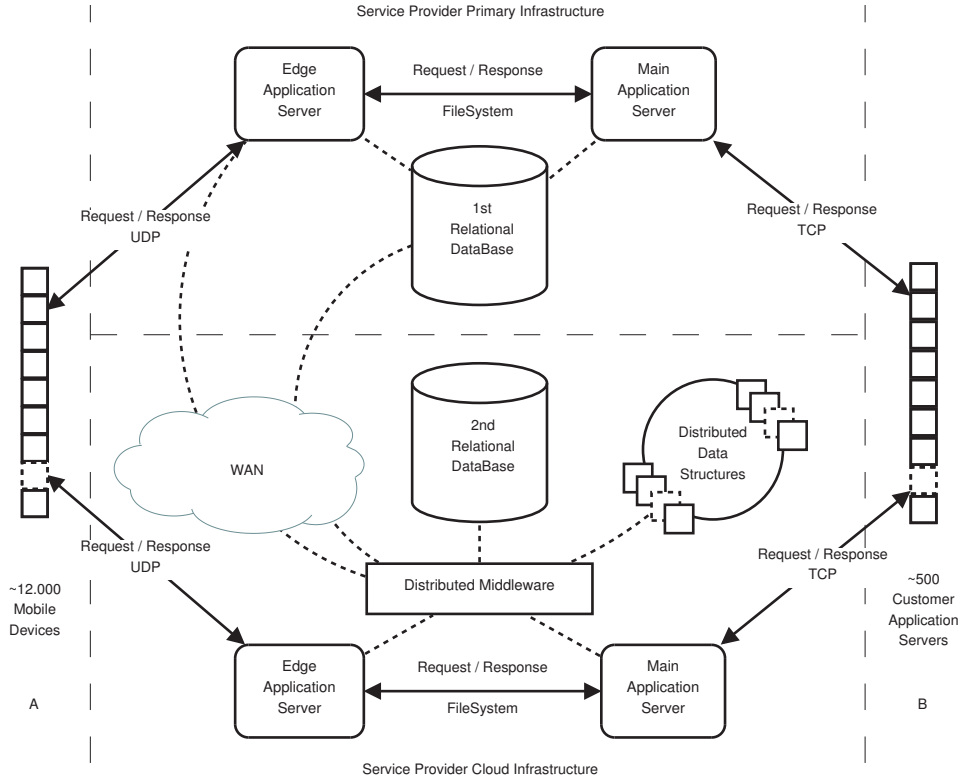

Figure 7. The improved environment, communicating with LIS over the WAN and making use of distributed middleware.

that is able to communicates with both customers and devices simulator. The requests set was chosen such that they fully cover the messages variety, which can occur in production.

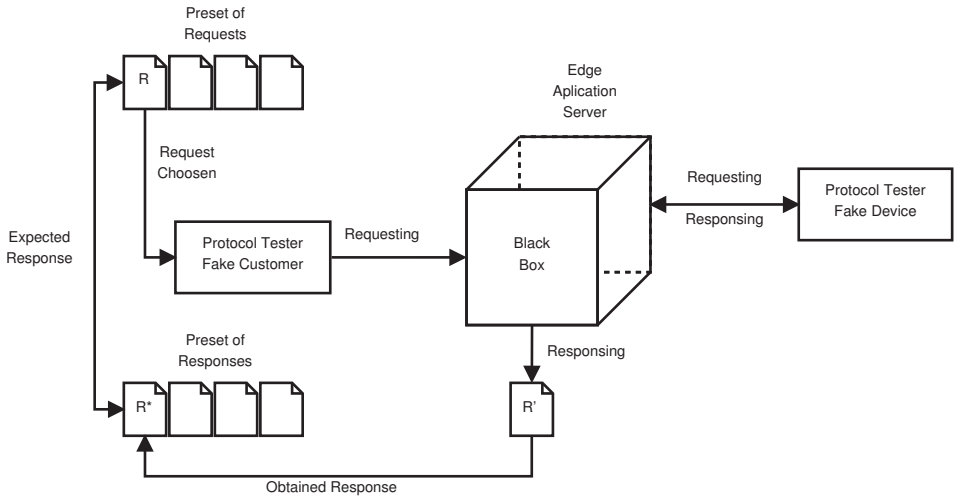

Figure 8. Devices and Customers simulation, handling EAS as a blackbox. The received responses set should match to the expected responses in order the codebase changes be accepted.

Each chosen request R, owns an expected $R*$ response, previously selected as a mean of acceptance. Once the protocol tester has selected any R message, then it is being fired against the blackboxed server, generating a response R' that is received and

compared with its related expected response $R*$.

If both R' and $R*$ are equals, it means that the developed feature does not affects negatively the server behavior and then, we can accept that codebase changes. However, whether the the comparison lead us to different results, the proposed changes must me rejected and an accurate analysis of R' could provide us traces on what is wrong with the evaluated implementation. Once these improvements were aggregated to the protocol testing process, we could ensure that all known behaviors would be retained while new features could be continuously added with security.

Since we developed a distributed solution with several daemons, we needed to ensure the tests process during the architectural changes conceiving could be agile. Thus, we developed a set of scripts that are able to setup a test environment with the applications full stack deployed into and ready to use. The main characteristics found in this script set which turned it into a reliable tool during codebase changing are:

- Ordered Startup and Shutdown: By means of configuration, we could set precedence between daemons, once there are productors and consumers in entire solution. Special parameters passing were allowed for bootstraping, such as activation of remotely monitoring on internal application states.
- Multiple hosts operations: Daemons can be activated and deactivated properly in different nodes along the network, according to test nature.
- Real-time log watching: Failures could be found by daemon alerts within log files, so when some miss behavior is detected upon previously stated patterns, we could setup stop conditions or else we can monitor for some expected behavior, in order to signal that whole operations are fully cohese and then could be continued.
- Ensure the full stack is properly running: It imposes the condition that no daemon could be solely ran. So, all the data flow is ensured and no end-point will fail to communicates with another ones due secondary reasons, as not being properly started.

6. Future Works

During the development of this work, we could state some points we believe to deserve more in depth attention. Thus, according to our past experiences it lead us to list them as potential contributions on the legacy and distributed systems fields of knowledge.

Considering the solution evolved in the sense of being able to operate with multiple instances of the fullstack, simultaneously, we suggest a measurement on how horizontal the solution could be scaled. We expect such rate reveal us to what extend the load applied over the solution could be relieved, simply by starting clone instances, so they can co-operate while doing load balancing themselves.

The auxiliary simulators that were developed, especially to migration process, could be extended to become generic enough, allowing its complete customization for another LIS testing purposes. This way, in the future they can be known as open tools for testing and protocols validation, enabling a wider range of blackbox tests on general purposes servers.

The set of scripts that handles startup and shutdown over the daemons, are found in a flexibility level enough such as it can be used in heterogeneous applications of several

kinds and, therefore, maybe it could be widely used by Platform as a Service, Cloud Computing based industry, because it allows the management of different services located in apparted hosts, building in a completely distributed way the requested setup.

7. Conclusion

The importance of this Case Study, was primarily concerned on addressing how a viable transition to a modern infrastructure could be made smoothly, while providing means for secure evolving of the whole solution, with its behavior fully validated on each iteration of migration steps, where brand new features could be developed and integrated to main codebase of the project.

The imposed fragility over the full operations chain, was essentially related to the stateful nature of critical data, that were in-memory stored. This phenomenon was successfully overcame after the aggregation of distributed middleware, which uses distributed data structures that resembles queues to handle the operations demand that made changes on RDB and memory states.

Conceptually, such data structures describe a query buffer such that they are fully compatible themselves, eliminating problems on storing batch operations into the memory from a single server, affecting them end-to-end in a non-transactional flow. Thus, even there are running operations, it could be immediately resumed by any EAS active instance, no matter what reason lead other service nodes to outages.

To ensure that EAS behavior has not been changed during the development and migration process, protocols testers were made featuring output validation by means of known and expected responses, attached to both ends of servers, which were able to be handled as different blackboxes on each planned migration step, securing that the solution could evolve without services disruption neither inclusion of bugs on its codebase.

References

Bisbal, J., Lawless, D., Wu, B., and Grimson, J. (1999). Legacy information systems: issues and directions. *IEEE Software*, 16(5):103 –111.

Brinkkemper, S. (1996). Method engineering: Engineering of information systems development methods and tools. *Information and Software Technology*, 38(4):275–280.

Brodie, M. and Stonebraker, M. (1995). *Migrating legacy systems: gateways, interfaces & the incremental approach*. Morgan Kaufmann series in data management systems. Morgan Kaufmann Publishers.

Canfora, G., Fasolino, A. R., Frattolillo, G., and Tramontana, P. (2008). A wrapping approach for migrating legacy system interactive functionalities to service oriented architectures. *Journal of Systems and Software*, 81(4):463 – 480. Selected papers from the 10th Conference on Software Maintenance and Reengineering (CSMR 2006).

De Lucia, A., Francese, R., Scanniello, G., and Tortora, G. (2008). Developing legacy system migration methods and tools for technology transfer. *Softw. Pract. Exper.*, 38(13):1333–1364.

Erradi, A., Anand, S., and Kulkarni, N. (2006). Evaluation of strategies for integrating legacy applications as services in a service oriented architecture. *IEEE International Conference on Services Computing*, pages 257–260.

Fayçal, H., Habiba, D., and Hakima, M. (2010). Integrating legacy systems in a soa using an agent based approach for information system agility. *International Conference on Machine and Web Intelligence (ICMWI)*, pages 338 –343.

Herrera, R. and Felinto, A. (2012). A distributed, multi-staged, high-throughput middleware for relational databases. *IIX Simpósio Brasileiro de Sistemas de Informação (SBSI)*.

Lewis, G. and Smith, D. B. (2007). Developing realistic approaches for the migration of legacy components to service-oriented architecture environments. *2nd International Conference on Trends in Enterprise Application Architecture*, pages 226–240.

Lu, F., Huang, H., Xu, Z., and Yu, H. (2005). A middleware for legacy application wrapper. *First International Conference on Semantics, Knowledge and Grid*, pages 47–.

Nasr, K. A., Gross, H.-G., and van Deursen, A. (2010). Adopting and evaluating service oriented architecture in industry. *14th European Conference on Software Maintenance and Reengineering*, pages 11–20.

O'Brien, L., Smith, D., and Lewis, G. (2005). Supporting migration to services using software architecture reconstruction. *13th IEEE International Workshop on Software Technology and Engineering Practice*, pages 81–91.

Parsa, S. and Ghods, L. (2008). A new approach to wrap legacy programs into web services. *11th International Conference on Computer and Information Technology (ICCIT)*, pages 442 –447.

Sneed, H. M. (1995). Planning the reengineering of legacy systems. *IEEE Softw.*, 12(1):24–34.

Umar, A. and Zordan, A. (2009). Reengineering for service oriented architectures: A strategic decision model for integration versus migration. *J. Syst. Softw.*, 82(3):448–462.

Vemuri, P. (2008). Modernizing a legacy system to soa - feature analysis approach. *IEEE Region 10 Conference (TENCON)*, pages 1 –6.

Warren, I. and Ransom, J. (2002). Renaissance: a method to support software system evolution. *26th Annual International Computer Software and Applications Conference (COMPSAC)*, pages 415 – 420.

Wu, B., Lawless, D., Bisbal, J., Grimson, J., Wade, V., O'Sullivan, D., and Richardson, R. (1997). Legacy systems migration-a method and its tool-kit framework. *Asia Pacific Software Engineering Conference and International Computer Science Conference (APSEC / ICSC)*, pages 312 –320.

Zhang, B., Bao, L., Zhou, R., Hu, S., and Chen, P. (2008). A black-box strategy to migrate gui-based legacy systems to web services. *IEEE International Symposium on Service-Oriented System Engineering*, pages 25–31.