A Lightweight Architecture Analysis of a Monolithic Messaging Gateway

Daniel Brahneborg
Infoflex Connect AB
Stockholm, Sweden
Email: brahneborg@infoflexconnect.se

Wasif Afzal
Mälardalens Högskola
Västerås, Sweden
Email: wasif.afzal@mdh.se

***Abstract*—*Background:* The Enterprise Messaging Gateway (EMG) from Infoflex Connect (ICAB) is a monolithic system used to deliver mobile text messages (SMS) world-wide. The companies using it have diverse requirements on both functionality and quality attributes and would thus benefit from more versatile customizations, e.g. regarding authorization and data replication. *Objective:* ICAB needed help in assessing the current architecture of EMG in order to find candidates for architectural changes as well as fulfilling the needs of variability in meeting the wide range of customer requirements. *Method:* We analysed EMG using a lightweight version of ATAM (Architectural Trade-off Analysis Method) to get a better understanding of how different architectural decisions would affect the trade-offs between the quality requirements from the identified stakeholders. *Result:* Using the results of this structured approach, it was easy for ICAB to identify the functionality that needed to be improved. It also became clear that the selected component should be converted into a set of microservices, each one optimized for a specific set of customers. *Limitation:* The stakeholder requirements were gathered intermittently during a long period of continuous engagement, but there is a chance some of their requirements were still not communicated to us. *Conclusion:* Even though this ATAM study was performed internally at ICAB without direct involvement from any external stakeholders, documenting elicited quality attribute requirements and relating them to the EMG architecture provided new, unexpected, and valuable understandings of the system with a rather small effort.**

***Index Terms*—Architecture, Monolith, ATAM, Microservice**

I. Introduction

Mobile text messages (SMS) are used world-wide, being popular as they work on all mobile phones without any additional software installed. In particular, they are commonly used by companies to send meeting reminders, authentication codes, travelling tickets, and more, to their customers. Between these companies and the customers' network operators, we find a product segment called SMS gateways. These gateways receive text messages from the companies, route them to the right operators, manage the connections to the operators over several different communication protocols, and handle any operator specific requirements. The gateways are often run by a separate group of companies, known as SMS brokers.

One of these gateways is the Enterprise Messaging Gateway (EMG) from Infoflex Connect (ICAB). EMG is a system with a proven track record spanning more than 20 years. Its monolithic architecture makes it easy for customers to deploy and manage EMG, and is also convenient for the software developers as the entire code base is just a simple function call away.

However, monoliths can be difficult to scale horizontally, i.e. to more than one server, and are sensitive to failures as those can bring the entire application down [1]. In our context, it is also problematic that each software update requires a full application restart, temporarily stopping all traffic.

One common way of addressing some of these issues with monoliths is to split them into sets of microservices [2], [3]. Selecting which parts of the monolith to extract is non-trivial [4], [5], but usually involves identifying components with loose coupling (independent) and strong cohesion (self-contained) [6].

Older literature on modularity provide some other recommendations, such as "*We propose instead that one begins with a list of difficult design decisions or design decisions which are likely to change. Each module is then designed to hide such a decision from the others.*" by Parnas [7]. We find a similar theme in many of the design patterns by Gamma *et al.* [8], namely to "*encapsulate the concept that varies*".

The approach used by Cruz *et al.* [9] was to base their migration of a monolith to microservices on the results of an architectural analysis of the system. Such an analysis focuses on aspects which arise as a consequence of the architecture, e.g. response times, scalability, and modifiability. In particular, they used the Architectural Trade-off Analysis Method (ATAM) created at the Carnegie Mellon's Software Engineering Institute [10], [11]. According to both Dobrica *et al.* [12] and Anjos and Zenha-Rela [13], the only analysis methods that consider a wide set of quality attributes are ATAM and SBAR [14]. Of these, ATAM is the only one applicable to mature products [13]. A more recent approach is RCDA, the Risk- and Cost Driven Architecture approach [15], adding a financial dimension to architectural work. RCDA appears to be most useful early in relatively large projects, neither of which is the case for us.

We know from experience that software such as EMG, created by one company and used by others, usually require variability in terms of customer specific behaviour. This variability is typically not required for microservices, as they tend to be created for in-house use and even operated by the developers themselves, sometimes phrased as "*you build* *it, you run it*" [16]. This in-house focus means that "*the literature [on microservices] is scarce in relation to the use of variability*" [2]. Variability is also not covered in the otherwise comprehensive mapping study on microservice architecture by Alshuqayran *et al.* [17].

The research questions we address in this work are whether ATAM could help identifying the components in EMG where architectural changes would be most beneficial, and whether it can help clarifying the need for variability in those components. During the analysis, all known quality requirements as given both by ICAB and their customers must be taken into account.

We present the results of the systematic application of a lightweight version of ATAM to ICAB's EMG system. Our primary contribution is showing that the ATAM analysis was able to identify the best component to change, which in ICAB's case is the credit management. Our second contribution is showing that ATAM also helped with managing the variability, suggesting that the credit management component should be extracted into a set of microservices, all providing the same API but implementing different strategies. Our third contribution is the overview of the ATAM artefacts and concepts (Fig. 1), with more details than the original paper [10]. Our fourth contribution is a description of how ATAM can be used in a real-world architecture analysis, even in the absence of external stakeholders (Section III-E).

The analysis was carried out as an entirely internal project at ICAB, avoiding the overhead that comes with involving external parties. While this setup presented a risk of missing some of the requirements and ignoring the relative importance of the requirements we found, it also meant that the study could be completed in just over one man month.

The current section has presented the background for the analysis and related work. Next, Section II describes ATAM and how we adapted it for our purposes, Section III presents the results from the ATAM analysis, and Section IV discusses the interpretation of those results. Section V discusses the validity threats and finally, Section VI presents our conclusions and planned future work.

II. Method

ATAM [10] is a systematic way to "assess the consequences of architectural decisions in light of quality attribute requirements". We therefore only consider requirements whose measurable responses to external stimuli are affected by the architecture.

An ATAM study follows the nine steps below, typically in a two or three day discussion workshop. The participants of this workshop are the various stakeholders of the system. The analysis is usually carried out in two phases, where phase 1 is limited to steps 1–6 and a small team, and phase 2 includes all steps a few weeks later, now with the full team.

Presentation:

1) Present ATAM to the participants.
2) Present the system from a business perspective.
3) Present the suggested architecture.

Investigation and analysis:

4) Identify architectural approaches. These approaches refer to aspects such as whether the system would be a monolith, client–server, or something else. Other terms used here are "styles" and "patterns".
5) Generate the quality attribute utility tree, a hierarchical list of quality requirements.
6) Analyze the architectural approaches in step 4, on how they realize the most important quality attributes described in step 5. This results in the identification of sensitivity points where an architectural approach affects a quality attribute, tradeoff points where an approach affects multiple quality attributes in different ways, and the list of such points that present a risk.

Testing:

7) Elicit and prioritize scenarios. Now the full set of stakeholders are included, brainstorming both current use cases, expected future "change scenarios", and extreme "exploratory scenarios".
8) Analyze the architectural approaches again, now focusing on the most important scenarios from step 7. If a scenario can not be realized using the selected architectural approaches, these need to be adjusted.

Reporting:

9) Present the results.

The key concepts used in ATAM, according to our understanding, are shown in Fig. 1. We noted that there is a discrepancy in the ATAM paper regarding the output from step 6. First it says "*all sensitivity points and tradeoff points should be categorized as either a risk or a non-risk*", suggesting that risk is an orthogonal concept to the first two. However, then it continues "*The risks/non-risks, sensitivity points, and tradeoff points are gathered together in three separate lists*", where the risks are now a concept of its own. In this paper we used the second variant, with the risks list containing attribute goals we do not yet know how to fulfill.

We made some additional adaptations of ATAM to fit our situation better. First of all, no external stakeholders were involved. Therefore, our analysis was based on the lightweight version of ATAM [11], where the testing done in steps 7 and 8 is skipped. This is, in essence, ATAM phase 1 plus reporting. The current section represents step 1 and Section III represents step 9, containing the output produced by steps 2 to 6.

Next, we realized that the prioritization of the quality attributes in step 6, "Analyze Architectural Approaches", was highly dependent on which stakeholders were present. To get an objective result despite the lack of such stakeholders, we needed another way of identifying the most important scenarios. For this purpose, we counted the number of business driver groups interested in each of the quality attributes. The point of having a varied set of stakeholders would partly be to have somebody represent all these different groups, making this simple count seem a reasonable proxy.

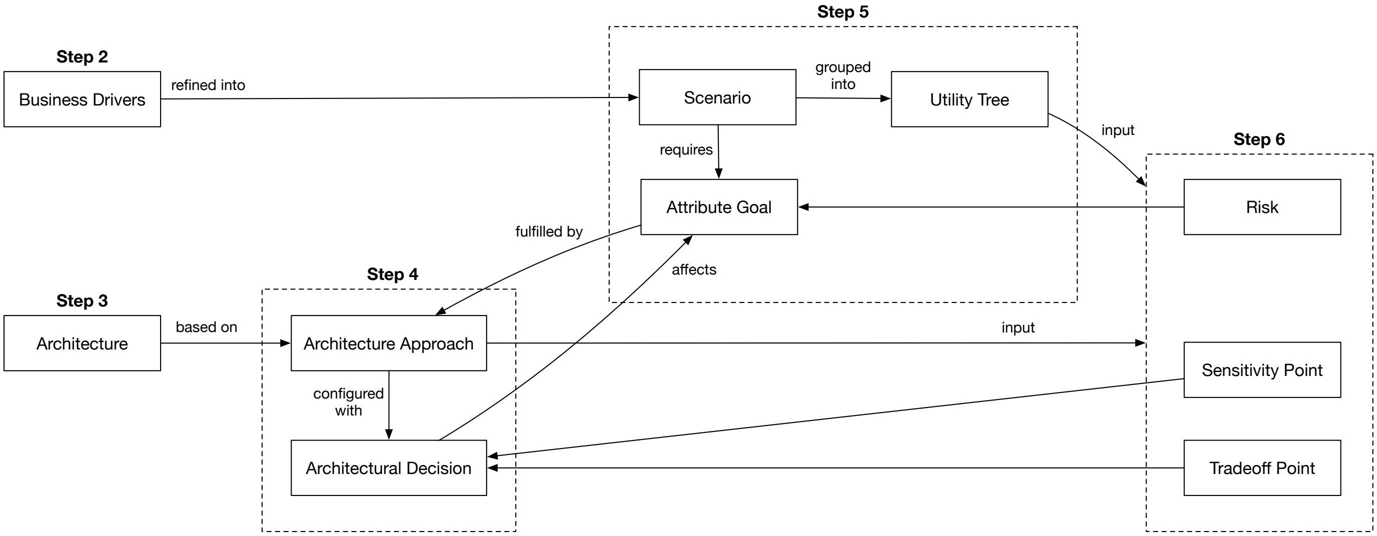

Fig. 1. Artefacts created by the different steps in the ATAM version used in this paper, with arrows indicating data flow. The analysis output is discussed in detail in Section III-E.

III. Results

This section represents step 9, presenting the output generated from steps 2 to 6. We recall that step 1 was covered by Section II, and that steps 7 and 8 were skipped entirely.

A. Step 2: Business Drivers

The system requirements from a business perspective were grouped into "important functions", "major quality attribute goals", "business goals", and "constraints".

- The most **important functions** of EMG are to:
  - forward messages with high throughput, and
  - manage client credits for these messages.
- The **major quality attribute goals** of EMG are to:
  - be available without interruptions,
  - prevent data loss, and
  - be easy to install and maintain.
- To satisfy the **business goals**, each EMG release should:
  - provide increased value to a varied set of both old and new customers, and
  - be done several times per year, to minimize risks for both ICAB and the SMS brokers by not containing too many new features or updated behaviours.
- The **constraints**, defining the border between what can and what can not be done, say that:
  - EMG must follow standard network protocols (e.g. SMPP and HTTP),
  - the relative message ordering does not need to be maintained [18], and that
  - ICAB does not have the resources for a full software rewrite. This means focusing on the smallest changes in the most isolated modules, giving the most value to the largest number of customers.

Step 2 also includes the identification of the major stakeholders, which in our case consists of three groups. The first group consists of the staff at ICAB, where there is a strong focus on maintaining a sound architecture, making the required effort for adding new features predictable. In the second group we have the entry level customers, operating gateways with a limited amount of traffic. These customers run EMG on a single node, trading potentially lower availability for a lower cost and a simpler setup. The third and final group consists of customers using multiple EMG nodes and complex configurations to achieve higher system-level throughput, higher availability to clients, and better protection of queued messages.

B. Step 3: Analyzed Architecture

At the highest abstraction level, EMG, installed at an SMS broker, sits between one or more clients on one side, and one or more mobile network operators on the other side. The clients send messages to EMG, the messages are routed within EMG according to the site specific configuration, and persisted on disk until they can be sent to the designated operator. After a message has been acknowledged by the operator, it is removed from disk. Additionally, the operator can send back message specific delivery reports, which are handled much the same way as normal messages, but travelling in the opposite direction.

To maximize the throughput whilst minimizing the complexity of the installation and configuration, EMG is a modular monolith. Optionally, customer specific plugins can be used at a handful of well defined points in the message life cycle.

EMG normally only needs to be restarted for installation of optional software updates, typically less than a few times per year. In order to prevent data loss and make these restarts as opaque as possible for the clients, it is critically important that the restarts are handled correctly. To accomplish this, the modules in EMG are grouped into three types based on how their data is managed. The modules without persistent state

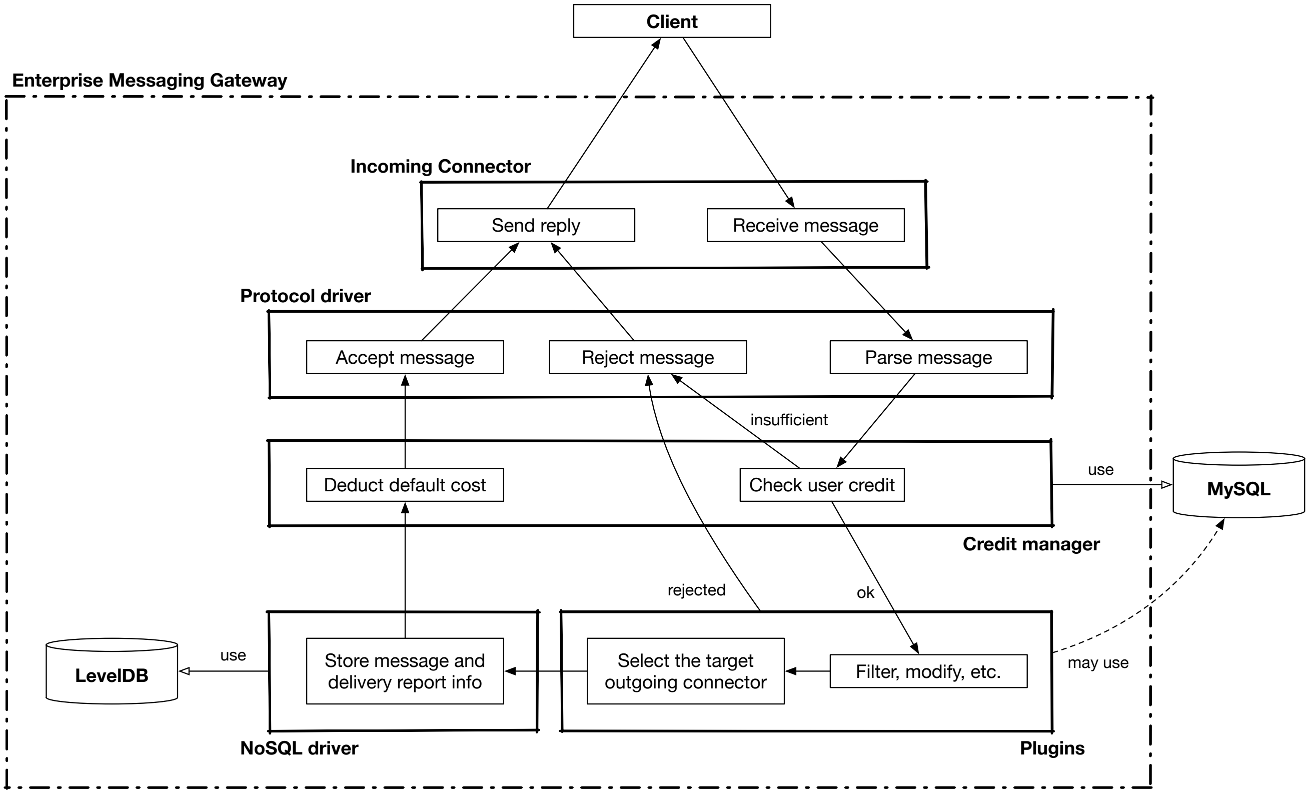

Fig. 2. The main data flow for incoming traffic from clients.

comprise the first type. The second type uses an embedded NoSQL database (currently LevelDB from Google), and the third type uses an external MySQL database.

Modules of the first type provide functionality such as network connectivity and protocol drivers (e.g. SMPP and HTTP), address and content filtering and modification, message routing, and logging. Most of these modules can use either configurable builtin logic or a site specific plugin, developed either by ICAB or the customers themselves.

The data managed by the second type of modules, using an embedded database, is not accessible from the outside. This enables ICAB to change both the actual database used as well as the structure of the stored data, as needed. These modules primarily handle the message queues and information about pending delivery reports. There is also a module for the SAT (Source Address Translation) functionality, providing a dynamic mapping between the incoming sender address from the client and the address used towards the operator. This functionality enables a client to use an email address for the sender address, which is mapped to a phone number picked from a number pool. The message recipient can then reply to this pool number, which the SAT module converts back to the original email address.

The third type of modules manage data using the MySQL client API, making the data available for modification from the outside. This gives SMS brokers the option to use either a simple single node database or a multi-node replicating cluster, without requiring any special handling by EMG. These modules mainly handle user authentication and message credits. There is usually also a read-only view of the last known state (e.g. received, forwarded, failed, or delivered) of each accepted message, used for billing and troubleshooting.

C. Step 4: Architectural Approaches

At the top level, EMG is best described as using the publish-subscribe [19] architectural style. Even though “connector” is part of the publish-subscribe style, in the EMG context it is used as an endpoint definition from clients or to operators. The embedded NoSQL storage acts as the event bus, the connection between producers and consumers. The publisher is driven by the incoming connector the client connects to, and the consumer is driven by the outgoing connector. The events are the text messages, and the event types correspond to the names of the connectors. Each event must only be received by a single consumer, to avoid multiple copies of each message appearing in the recipients’ mobile phones, barring exceptional circumstances.

Both producers and consumers use a simple layered approach. The data flow of the producer side is shown in Fig. 2, where each received message passes down through various modules before a response propagates back to the sender. The incoming connectors receive messages using one of the protocol drivers, filter, modify and route them as configured, and finally persist them in the NoSQL storage. Similarly, the outgoing side handles connectivity to the systems downstream,

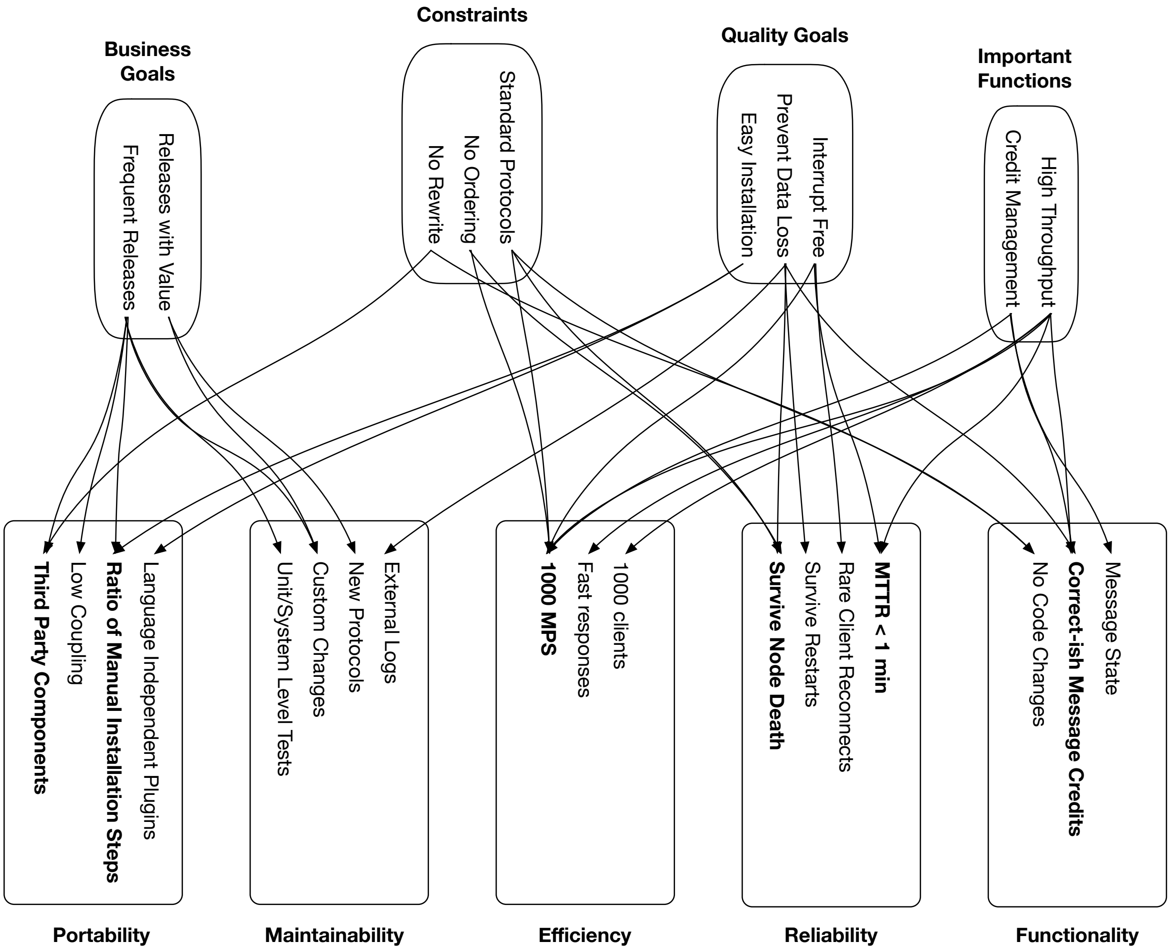

Fig. 3. Business Drivers (top) and Quality Attributes (bottom) for EMG.

forwards the messages, removes them from storage, and finally adjusts the user credit value. The usage of an embedded database provides good performance and protection from temporary node failures, but not from permanent node failures.

EMG is deployed on one or more, physical or virtual, 64 bit Linux nodes. The deployment requires the installation of a number of third party components, adding the executable files for EMG, and making site specific configurations. MySQL is deployed on the same or other nodes.

Using multiple EMG nodes is beneficial despite the lack of data replication between them, as permanent node failures are rare. Separate sets of messages can then be processed in parallel by all nodes, resulting in increased system-wide throughput.

Based on the assumption that the data in the embedded databases is limited in size, all such data is loaded on startup into specifically tailored in-memory data structures. Primarily, this frees the “find the next message to send” and “find the delivery report record corresponding to a given message id” operations from having to make time consuming round-trips to the database. These operations can now be carried out much faster by instead traversing the data structures.

D. Step 5: Quality Attribute Tree

In step 5, the quality attributes are elicited, prioritized, and refined with exact stimuli and falsifiable responses into scenarios, which are to be analyzed in the next step. Typically these attributes originate from the business goals, the result from step 2. ATAM provides a sample list of possible quality attributes, but also notes that stakeholders “*may add their own quality attributes or may use different names for the same ideas*” [10]. Another attribute list for use in ATAM is presented by Bass *et al.* [11]. We decided to base our list on SIS-ISO/IEC 9126 [20], because it had a focus on measurable attributes.

Defining all scenarios fully was not considered meaningful in this paper. For example, the functional requirement that all incoming messages should be forwarded can be refined with the quality requirement that this should be done as soon as possible, which in turn could be refined by setting a maximum time limit. However, the time a message is spent queued within EMG depends entirely on the availability of and throughput to the receiving side, both of which are beyond our control.

The full list of elicited quality attributes for the current use cases are summarized together with their corresponding business drivers in Fig. 3, and described in more detail below.

The attributes selected for step 6 are written in boldface.

1) Functionality: The database containing the current state of each message should be kept reasonably up-to-date. As this requirement was not among the ones analyzed in the next step, the exact limit for the allowed delay was not specified further.

In order for the EMG owner to be able to bill the clients correctly, and possibly also ensure messages are pre-paid, the current value of the message credits for each user must be kept up-to-date. A discrepancy of 1 second's worth of messages would be acceptable, if that increases the throughput.

Ever since EMG was created, it has followed the general robustness principle of being as tolerant as possible when processing incoming data, and as conservative as possible when producing outgoing data. Following this principle ensures a minimum of code changes are required for EMG to exchange messages with other systems, and has served the product well.

2) Reliability: There are two types of quality attribute requirements in the reliability category. The first type concerns the availability to clients, stating that forcing clients to reconnect to EMG should be a rare event, and happen at most once per several months. Additionally, when such an interruption occurs, for example due to EMG being restarted, connectivity should be restored in less than one minute.

The second type concerns the prevention of message loss. As there is no dependable end-to-end acknowledgement for text messages, a message received and confirmed back to the sender must remain in the queue and eventually get sent, regardless of whether the EMG application stops temporarily or permanently. The latter is of course only possible to achieve in a configuration with multiple nodes and data replication.

3) Efficiency: Most modules in EMG are I/O bound, either for disk, network, or both. In order to effectively utilize modern hardware with fast multi-core processors, each EMG node must support traffic from at least 1000 parallel client connections. Moreover, a majority of clients send very few messages, which means a large number of clients are required for the SMS gateway node to be profitable.

Next, incoming messages should result in an acknowledgement sent back within 10 seconds. Some SMS gateways wait up to a minute before sending back a reply, making it difficult for the sender to know whether the remote system is still operational. An EMG running on a local node has no problems replying within 1 millisecond, but this time increases significantly if routing is done using a plugin requiring one or more network roundtrips to other systems, or if the message must be replicated to a different data center.

Finally, each EMG node should be able to process at least 1000 messages per second (MPS). This number is based on both the requirements from existing EMG customers, and what seems to be reasonable given the current architecture. The highest throughput is typically reached for 10–100 parallel connections, as this keeps all processor cores busy.

4) Maintainability: For maintainability, where the requirements are based almost entirely on the business goals of doing frequent and valuable releases, we consider both development and operational perspectives. For the development perspective, adding new protocols and custom logic should not affect the rest of the system. This is supported by a comprehensive regression test suite. For the operational perspective, the most important requirement is to assist troubleshooting by having EMG log over the network.

5) Portability: The portability requirements concern the independence of the system from its surrounding environment, and of its parts relative to each other. To achieve the former, plugins should be possible to implement in any programming language available on the Linux platform, and the number of manual steps in the installation procedure should be kept to a minimum. Until recently, plugins could only be written in C and Perl, but with the recent addition of an HTTP interface they are now language independent. For the installation, having to install multiple system packages can cause conflicts with other applications. Achieving the latter, where the parts are independent from each other, enables making isolated and predictable changes. Keeping system parts independent is easier when using third party components, as they are guaranteed not to have any dependencies back into EMG.

E. Step 6: Analyze Architectural Approaches

In step 6, the architectural approaches from step 4 are analyzed on how well they support the most important requirements from step 5. As mentioned, we found those requirements by counting the number of business goals they were related to. This way we found "1000 MPS" linked to 3 groups, followed by "correct-ish message credits", "MTTR $<$ 1 min", and "survive node death", each one linked to 2 groups. We also found links to 2 groups from "ratio of manual installation steps" and "third party components". We recall that these requirements are all marked with boldface in Fig. 3. Conversely, each one of the business drivers "high throughput", "prevent data loss" and "standard protocols" affects the largest number of quality attribute groups, i.e. 3. We describe the most significant architecture parameters related to these attributes below.

1) Risks: As motivated in Section II, our list of risks contains "*architecturally important decisions that have not been made*" [10]. In our case, the only identified risk concerns the temporary storage of unsent messages. How to ensure messages survive the permanent failure of the node they arrived to, while still maintaining the desired throughput and following the constraints from step 2, is an open issue.

2) Sensitivity points: The sensitivity points are "*parameters in the architecture correlated to measurable quality attributes*" [10]. Production environment log files from various sites clearly show that the greatest effect on the time required to restart the EMG application after a failure, i.e. the MTTR, is due to the preloading of NoSQL data. The lion's share of the startup time is used loading the runtime data for the SAT functionality. This is because the SAT data is kept for several days, whereas messages waiting to be sent and data on pending delivery reports are typically removed after just a few seconds.

The ratio of manual installation steps and the coexistence with third party components would both benefit from improved installation procedures. In particular, using container technology such as Docker[1] could have considerable positive effects.

3) Trade-off points: Trade-off points are similar to sensitivity points, but "*correlated to multiple quality attributes with different effects*" [10]. The architectural decision of storing the message credits in an external database is the source of two such points, both related to performance.

The first point concerns the trade-off between performance and usability. By keeping the credits in MySQL, we improve usability as the credits are easy to read and update from an external tool. However, because each database operation takes non-zero time, at the same time we also lose throughput.

The second point is the trade-off between performance and precision. To mitigate the lowered throughput from the previous trade-off, credit updates could be limited to once per n messages. This would improve performance on account of fewer database operations, but reduce the precision of the current credit value. The credit value would be only eventually consistent at each point in time, so there are windows of time in which clients might be able to send more messages than they should be, according to their pre-paid message credits.

IV. Discussion

Using multiple EMG nodes can not only protect from data loss when an individual node fails, but also increase the total system throughput as more work can be done in parallel. Only the data protection would benefit from replication of the data in the embedded databases, in particular the messages yet unsent. In contrast, both data protection and throughput would benefit from having an effective way of keeping the message credits synchronized among all nodes, as it would allow clients to simultaneously connect to multiple EMG nodes. Furthermore, the ability of doing batch-wise updates of the message credits may provide increased throughput also for the smaller customers using a single EMG node.

We see that improving the credit management would have the biggest impact as it would increase the system throughput, and thereby the value of EMG, for basically all customers. Such an enhancement could be carried out in multiple steps, with each step producing a new variant to be released and then maintained independently. These variants would probably include, but not necessarily be limited to, the ones listed below. Fig. 4 shows how they replace the "use" arrow between the credit manager and MySQL to the right in Fig. 2.

1) A minimal microservice which checks and updates the credit value for every message, in the same way as the existing implementation.
2) A microservice which updates the credit value with a configurable frequency.
3) A microservice replicating the message credits between EMG nodes, allowing for a more efficient solution with higher throughput than with a replicated database.

The modularity of EMG frees the rest of the system from having to care whether credit updates are batched, if the values are replicated, and if so, how. The components not managing the credit value need only be able to ask if a user can send more messages, and request credit updates when the messages are processed. This loose coupling makes the credit management an excellent candidate for extraction to microservices. As a microservice it can be updated without violating the reliability requirement of allowing clients to remain connected to EMG for extended periods of time. Due to the increased operational complexity of microservices compared to a single monolith, ICAB may also consider implementing the batching of the credit updates as an optional feature in the EMG core.

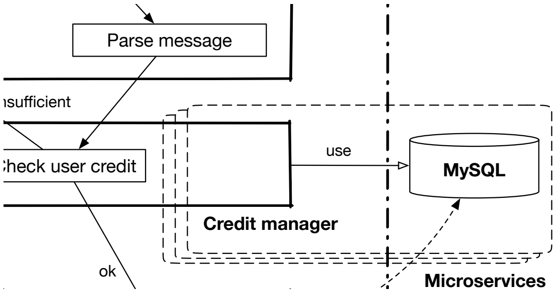

Fig. 4. The connection between the credit manager and MySQL, shown to the right in Fig. 2, would be replaced by one of three microservices.

The answer to our question regarding whether ATAM could help identifying the components in EMG where architectural changes would be most beneficial, is therefore a definite yes. It is also clear that ATAM can help clarifying the need for variability in such components. Additionally, the analysis showed the significance of the SAT implementation. The SAT feature is used by only a few EMG customers, but as the long startup time of the current implementation risks violating the important availability requirement, it needs to be updated.

A. Lessons learned

The EMG team members are well acquainted with the customer requirements since they all have been involved with the development of the product during its entire lifetime, spanning more than 20 years. The new insights provided by ATAM therefore came a surprise, as the extraction of the credit management into a separate component had never before been considered.

The flexibility of ATAM turned out to be very beneficial to us. As mentioned in Section III-D, the examined quality attributes could easily be adjusted for the specific system being analyzed. Despite ATAM being presented as a tool for improving the communication between stakeholders [10], we were able to adjust it to give meaningful results in ICAB's context even though none of the external stakeholders participated.

We were also able to identify prioritized quality attribute scenarios directly in Fig. 3 as described in Section III-E, without having to create an explicit quality attribute tree. This worked in our context as the focus was on highlighting the most prioritized scenarios, rather than making a finer prioritization in terms of "high", "medium" and "low". We appreciate that this allowed us to save both time and effort

[1] `https://www.docker.com`

as utility tree generation is known to be an effort consuming activity [9].

One additional benefit of applying ATAM in our context was the capture of variability scenarios, whereby the component undergoing architectural change would be customized according to the needs of different customers. This explicit identification of variability in architecture perfectly matches ICAB's business goals, and other companies can very well benefit from eliciting such variations without having to go all the way to CBAM (Cost-Benefit Analysis Method) [11], [21].

V. Threats to Validity

Runeson and Höst [22] have grouped validity threats into construct, internal, external and reliability. External validity threats concern whether the results are still valid in a more general context. We recognize that some of the simplifications mentioned in Section IV-A fall into this category.

Reliability threats concern whether other researchers would get the same result. The most problematic threat here is the list of business drivers elicited in step 2, and thereby also the list of quality attribute requirements analyzed in step 6, which may both be incomplete. We addressed this by revisiting the lists over several months, adjusting them until they stabilized.

Another reliability threat is the selection of attribute requirements from step 5, where we used a different method than suggested by ATAM. Given what we know about the existing EMG installations, the results still appear reasonable.

VI. Conclusions and Future Work

The monolithic application EMG needed to provide higher availability and better data protection, which motivated us to perform an ATAM analysis of the existing architecture. EMG is a messaging gateway, which stores and forwards short text messages, and is used by customers with very diverse requirements. As messages are such an important core concept of the application, we had long thought that the replication of those messages would be the next step forward for EMG. Taking all requirements into account, it instead became clear that updating the credit management, used for the billing of the senders of text messages, would provide the most value to the largest set of customers. This improvement will be realized by moving the credit management to a set of microservices, which in turn will allow each customer to select the balance between usability, performance and precision which best suits their particular needs.

For future work, we plan to evaluate the new credit management microservices on relevant quality attributes, e.g. code complexity and performance. ICAB also plans to investigate how best to improve the installation procedures, possibly by making use of container technologies.

Acknowledgments

This work was sponsored by The Knowledge Foundation industrial PhD school ITS ESS-H, Infoflex Connect AB and H2020 project ADEPTNESS (871319).

References

[1] D. Taibi, V. Lenarduzzi, and C. Pahl, "Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation," *IEEE Cloud Computing*, vol. 4, no. 5, pp. 22–32, 2017.

[2] L. Carvalho, A. Garcia, W. K. G. Assunção, R. Bonifácio, L. P. Tizzei, and T. E. Colanzi, "Extraction of Configurable and Reusable Microservices from Legacy Systems: An Exploratory Study," in *International Systems and Software Product Line Conference*. ACM, 2019.

[3] J-M. Horcas, M. Pinto, and L. Fuentes, "Software Product Line Engineering: A Practical Experience," in *International Systems and Software Product Line Conference*. ACM, 2019.

[4] J. P. Gouigoux and D. Tamzalit, "From Monolith to Microservices: Lessons Learned on an Industrial Migration to a Web Oriented Architecture," in *2017 IEEE International Conference on Software Architecture Workshops*. IEEE, 2017.

[5] J. Fritzsch, J. Bogner, A. Zimmermann, and S. Wagner, "From Monolith to Microservices: A Classification of Refactoring Approaches," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2019, vol. 11350 LNCS, pp. 128–141.

[6] S. Newman, *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith*. O'Reilly Media, 2019.

[7] D. L. Parnas, "On the Criteria to be Used in Decomposing Systems into Modules," *Communications of the ACM*, vol. 15, no. 12, pp. 1053–1058, 1972.

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

[9] P. Cruz, H. Astudillo, R. Hilliard, and M. Collado, "Assessing Migration of a 20-Year-Old System to a Micro-Service Platform Using ATAM," in *IEEE International Conference on Software Architecture Companion*. IEEE, 2019.

[10] R. Kazman, M. Klein, and P. Clements, "Method for Architecture Evaluation," Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst, Tech. Rep., 2000.

[11] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice, 3rd ed.* Addison-Wesley Professional, 2013.

[12] L. Dobrica and E. Niemelá, "A Survey on Software Architecture Analysis Methods," *IEEE Transactions on Software Engineering*, vol. 28, no. 7, pp. 638–653, 2002.

[13] E. Anjos and M. Zenha-Rela, "A Framework for Classifying and Comparing Software Architecture Tools for Quality Evaluation," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6786 LNCS, no. PART 5, pp. 270–282, 2011.

[14] P. Bengtsson and J. Bosch, "Scenario-Based Software Architecture Reengineering," in *5th International Conference on Software Reuse*. IEEE, 1998.

[15] E. R. Poort and H. Van Vliet, "RCDA: Architecting as a risk- and cost management discipline," *Journal of Systems and Software*, vol. 85, no. 9, pp. 1995–2013, 2012.

[16] J. Gray, "A Conversation with Werner Vogels," *ACM Queue*, vol. 4, no. 4, pp. 14–22, 2006.

[17] N. Alshuqayran, N. Ali, and R. Evans, "A Systematic Mapping Study in Microservice Architecture," in *IEEE International Conference on Service-Oriented Computing and Applications*. IEEE, 2016.

[18] D. Brahneborg, W. Afzal, A. Čaušević, and M. Björkman, "Towards a More Reliable Store-and-forward Protocol for Mobile Text Messages," in *2018 Workshop on Advanced Tools, Programming Languages, and PLatforms for Implementing and Evaluating Algorithms for Distributed systems*. ACM, 2018.

[19] G. Fairbanks, *Just Enough Software Architecture: A Risk-Driven Approach*. Marshall & Brainerd, 2010.

[20] ISO/IEC JTC 1, "SIS-ISO/IEC TR 9126-2:2003 Software Engineering – Product Quality – Part 2: External Metrics," Institute, Swedish Standards, Tech. Rep., 2003.

[21] R. Kazman, J. Asundi, and M. Klein, "Making Architecture Design Decisions: An Economic Approach," Carnegie Mellon Software Engineering Institute, Tech. Rep., 2002.

[22] P. Runeson and M. Höst, "Guidelines for Conducting and Reporting Case Study Research in Software Engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2009.