IntelliSMS Case Study - Building a Heavy Load Messaging System

About SoftwareMill

SoftwareMill, a software house based in Poland and Typesafe Consulting Partner was founded in 2009 to deliver high-quality, customized software solutions.

Our clients come from the Telecommunications, Banking, Logistics and Entertainment industries. A wide range of systems can be found in our portfolio, starting from high-performance mobile application backends, through fault-tolerant messaging gateways, big-data reporting solutions and credit management systems.

We take care of developing projects end-to-end, always trying to keep in mind that we want to develop maintainable, working software which brings value to our clients.

About IntelliSMS

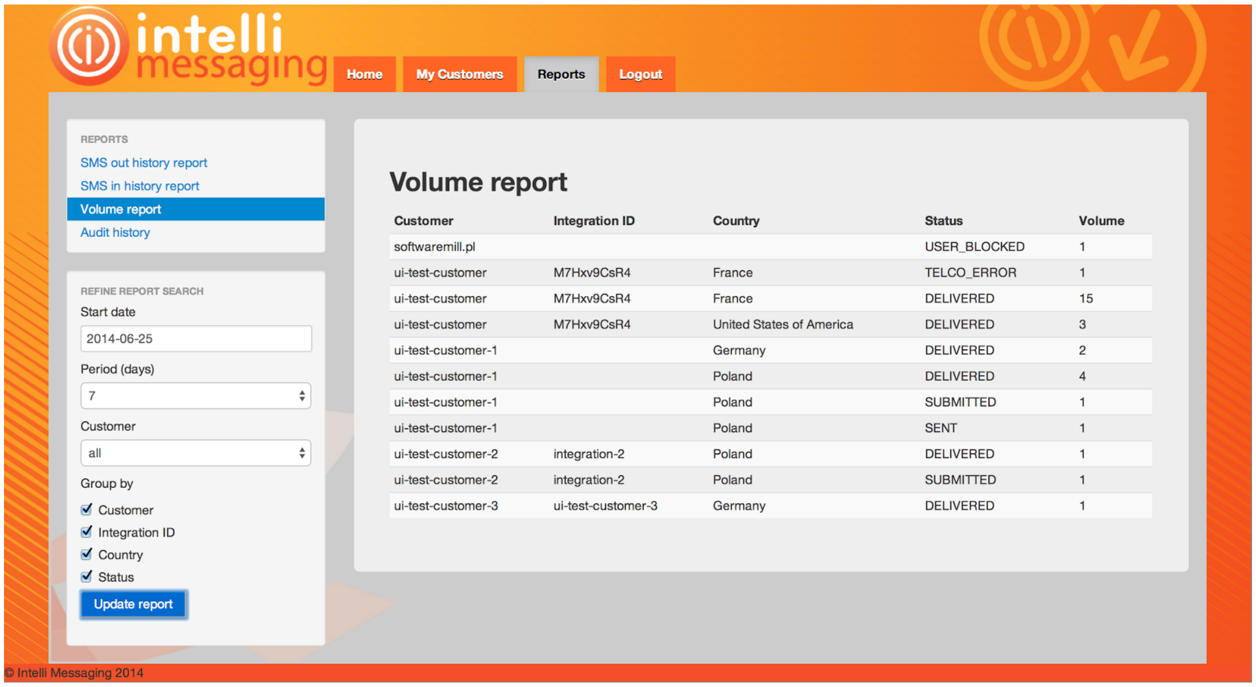
Intelli Messaging simplifies mobile communication methods so you can cost effectively build mobile communication into your business processes; Marketing or Operations. Whether you are an application provider, telecommunication services company or a large business we have the service and application offerings to support your needs.

The problem

IntelliSMS is present on the SMS market for a long time, since the year 2000, and through the years evolved to offer a wide range of SMS-related services:

- a wholesale enterprise SMS gateway,
- bulk and individual SMS sends,
- account and billing management for resellers,
- REST API for integrating with applications,
- 2-way SMS - replying to unique messages,

and many others. A couple generations of the MessageCore platform, backing the above mentioned services, were developed. The previous system, MessageCore4 (MC4), worked well in the beginning, but as more and more customers signed up and message volumes increased scalability problems started to surface, which hindered the business. Also, the code base grew large and hard to maintain, due to a number of features being added and removed over the years.



Volume report

Customer	Integration ID	Country	Status	Volume
softwaremill.pl			USER_BLOCKED	1
ui-test-customer	M7Hxv9CsR4	France	TELCO_ERROR	1
ui-test-customer	M7Hxv9CsR4	France	DELIVERED	15
ui-test-customer	M7Hxv9CsR4	United States of America	DELIVERED	3
ui-test-customer-1		Germany	DELIVERED	2
ui-test-customer-1		Poland	DELIVERED	4
ui-test-customer-1		Poland	SUBMITTED	1
ui-test-customer-1		Poland	SENT	1
ui-test-customer-2	integration-2	Poland	DELIVERED	1
ui-test-customer-2	integration-2	Poland	SUBMITTED	1
ui-test-customer-3	ui-test-customer-3	Germany	DELIVERED	1

That’s when IntelliSMS turned to SoftwareMill to take part in developing the next generation system, MC5. The core requirements were to create a scalable architecture, which would allow to accommodate for increasing volumes of messaging.

Another important requirement was for the messaging to be reliable. A large part of the messaging traffic is alert messaging, so it is vital that the messages are not lost. Combining a replicated, persistent message storage, which will deliver messages even in case of a downstream service not being available or a server crashing permanently, with the scalability and performance requirements, provided a challenging task for our team to work on.

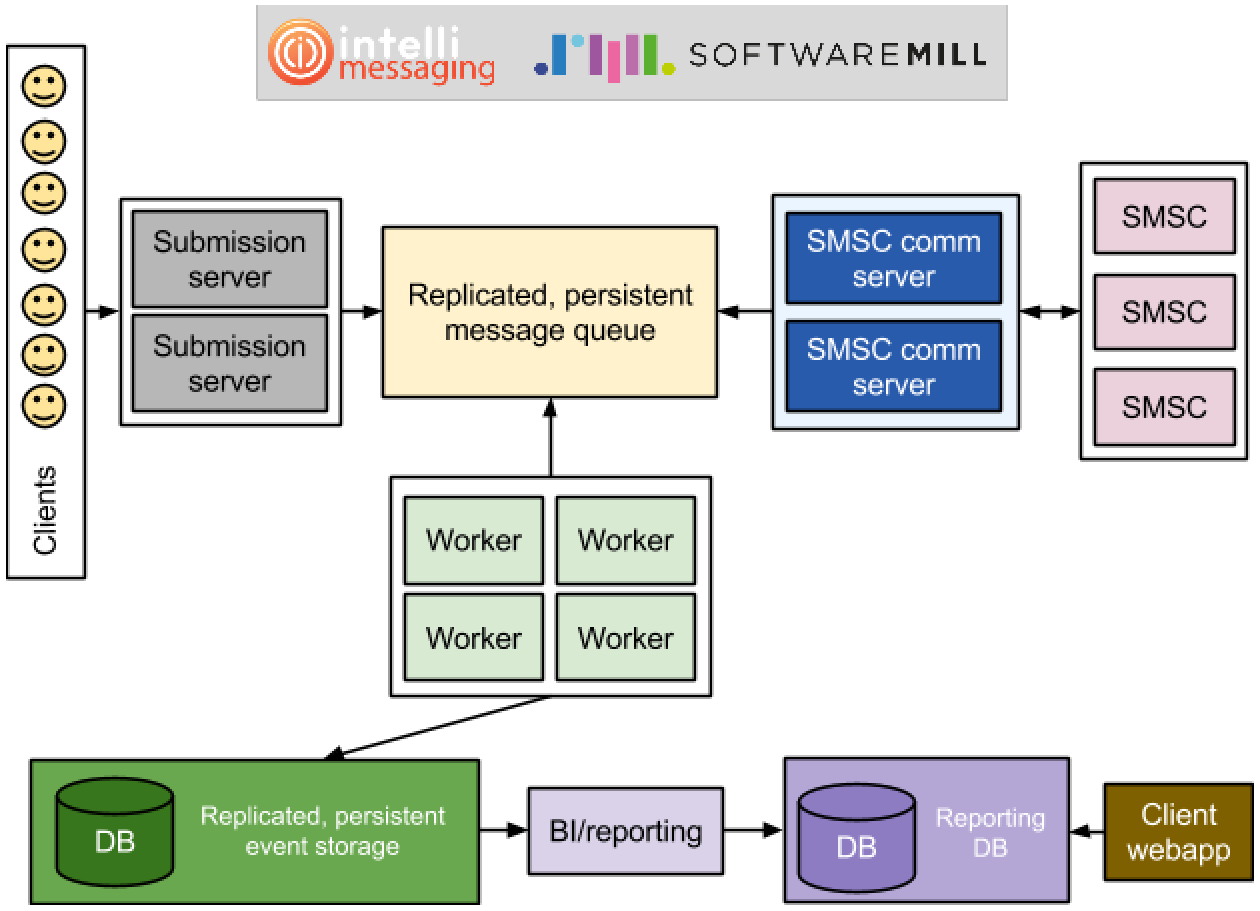
When creating the new system, an important factor was to maintain a clean, extendable code base, where new features can be implemented and plugged in relatively quickly, without a need to alter existing code in a significant way.

The solution

We started by drafting a system architecture. While we are strong believers in the Agile methodologies of developing and delivering software, and we try to minimize the amount of upfront planning and design, some planning is always necessary, to roughly guide the development. A good architecture allows the team to meet the basic system requirements, while being flexible enough to allow specific design decisions to be made as late as possible.

The architecture of MC5 assumed dividing the system into a number of independent components:

- a submission server, where users submitted sms send request, and queries for sms status
- workers, which through a series of pluggable handlers provided message routing, billing, reacting to reply SMS messages
- SMSC communication servers, which send the messages downstream to specific SMS providers
- reporting servers, to provide statistics and insights into SMS sending/delivery pattern



Each component is independent, can be scaled independently, and doesn't have to work at the same time as the other components. This is especially important for the SMSC communication servers, which connect to third-party providers; such links are inherently unreliable, and can go up and down unpredictably.

The components are written using pure Java; some of them are deployed in a servlet container, but most as stand-alone, fat-jars, which allows for easy management and fast provisioning. This adds to the simplicity of the system, which results in a lower operations and maintenance overhead.

The components communicate asynchronously through a persistent, replicated message queue. The queue implementation is pluggable, and currently we are using a custom solution based on MongoDB. As we are also using MongoDB for other purposes, and as MongoDB is great both because of the ease of installation, use and performance for streaming, replicated operations, it was a natural choice.

Our Mongo-based queue easily handles thousands of persistent, replicated messages per second, and can be scaled using the Starling/Kestrel model, by adding new sets of replicated nodes and using a random set of nodes when sending/receiving messages.

Apart from messaging, the system needs to persist SMS information at each stage of processing the message: when the SMS is accepted for sending, routed, delivered, and finally billed. This forms a natural stream of events, which get persisted to a Mongo collection. Not only we are able to handle very large volume of events efficiently (each event is immutable, so the Mongo collection is essentially append-only), but we get a clear audit trail for each message, in case we need to debug the system behavior.

Finally, the reporting system uses SQL database, which is populated using event sourcing from the Mongo collections. This can also be done efficiently, thanks to how Mongo works and the immutable, streaming nature of events. An SQL database was chosen for reporting because of the familiarity of the system users with the language and the flexibility in forming queries, and getting responses quickly.

Designing for performance and scalability is one thing, making sure the system is actually performant and scalable is another. That is why one of the first tasks we completed, after a skeleton of the system was done, was setting up an overnight, automated stress test, where the system was sending and receiving simulated SMS messages over 4 hours. This let us iron out many integration issues, on the Operating System, Load Balancer, Mongo and application levels. As the test runs each day, after a day of work we quickly know when we have degraded the system's performance.

As mentioned in the introduction, code quality and maintainability was key when developing the project. We are using a number of now standard methods, like (fast) unit and (slower) integration tests, adhering to "clean code" rules, paying attention to naming, keeping classes small, with a single responsibility and so on. Each change is also code-reviewed by other team members, both to catch bugs and design flows, but also to spread the knowledge about the system.

Remote project delivery

But technology isn't everything. Equally, if not more important is the way teams collaborate to deliver software. As IntelliSMS is based in Melbourne, Australia, and SoftwareMill in Warsaw, Poland, we formed a truly distributed team (SoftwareMill is additionally a fully distributed company, our employees come from many parts of the country), working remotely across two quite different time zones.

We tried to keep the formalism of our cooperation as low as possible. When it seemed beneficial, we use tools such as TinyPM or Confluence. We had no strict requirement documents, instead we tried to communicate as frequently as possible to deliver what will be the highest value for IntelliSMS, and hence the highest value for IntelliSMS users. That way we could adapt quickly based on the continuously gathered feedback.

Moreover, at SoftwareMill we have a number of proven methods for effective asynchronous and synchronous communication in a distributed setup. In the end the fact that we work remotely was barely noticeable.

The result

The system is in production for almost two years now; it has proved to work well under load, and the architecture met the original requirements. The code base is kept in a good shape, allowing existing developers to modify the code quickly and new developers to join the project without problems.

We have delivered a performant and resilient SMS platform, which can be scaled to meet future traffic demands, and evolved to include new features, to meet emerging business needs.

With a very difficult development task with some very high benchmarks SoftwareMill has delivered a great result and done so very cost effectively.
Peter Humphries,
Executive Director at Intelli Messaging