Week 6: Examine Architectural Tactics

Nate Bachmeier

TIM-7140:Software Engineering

April 4, 2021

Northcentral University

# Architectural Tactics

An architectural tactic is a design pattern that seeks that efficiently address a business problem. When software does not correctly implement these tactics, it produces significant risk toward the application’s security, reliability, and maintainability. Mirakhorli (2014) states once the architect becomes a “pile of mud,” any future changes likely come with a high regression risk. For instance, a program might hardcode the credentials or follow a monolithic design. These anti-patterns prevent the software from evolving into future needs. It is therefore critical that engineers understand design decisions and build to specification.

# Literature Review

Kontopoulos et al. (2018) describe the challenges of validating Automatic Identification System (AIS) metadata. Ensuring the accuracy of this information is critical for safety reasons. However, making that determination is complex due to the data’s volume.

## Problem Statement

Shipping vessels use this insecure protocol to transmit their location and intended route. The initial design requires that only ships exceeding 299 tons send these messages. However, countless smaller boats voluntarily leverage the system for safety purposes. Due to the data’s high volume, performing analysis is overly complex. Fraudulent and malicious transport companies spoof their location to exploit these issues. Along with hiding their location, it is also possible to fabricate congestion, causing legitimate cargo vessels to divert.

## Problem’s Importance

Addressing this big data problem is essential for preventing illegal activity, ensuring efficient routing, and protecting sailor’s safety. Exploiting the AIS data enables scenarios, such as illegal fishing and whale hunting. Farming animals into extinction will create irreversible changes to the fragile wildlife. Malicious actors that redirect third-party ships can increase costs and delays, creating unfair competitive advantages. When participants lose confidence in a safety system, that will result in accidents. These risks justify research investments into detecting deceptive activity and preventing it.

## Research Approach

Kontopoulos et al. (2018) propose a scalable architecture for processing AIS firehose. Their solution begins with AIS messages flowing into the master node. After decoding the payload, a load balancer routes the traffic to an appropriate worker node. Message routing attempts to reuse the same worker node for repeated vessel updates. This decision improves caching performance, not calculation accuracy. After determining the ship’s delta position, a central consensus node predicts the probability of being spoofed. The prediction uses historical information to learn across many features like speed and routing consistency. The authors base their predictions on a combination of machine learning and statistical tests. These assessments include Support Vector Machines and feature deviations from the mean.

## Result Verification

The authors determine the system’s accuracy by using 2.4GB of historical data from May 2016. After training the model, they use varying levels of data fuzzing to simulate malicious locations. These results show their solution has a precision above 85% and recall above 90%. There are specific classes of false positives that require further analysis. For example, flooding the worker nodes with too many wrong messages produces concept drift. Addressing those enhancements was outside the project’s scope. However, querying an external source of truth such as the manufacture design specifications could mitigate these issues. It would also be possible to learn these norms by examining more extended periods.

## Result Appropriateness

Kontopoulos et al. (2018) could go one step further by working with external maritime agencies to verify known malicious actors. This verification step would further confirm the detection mechanisms align with historical cases. Without that information, data fuzzing is an efficient approach to cover both known and unknown scenarios. This capability derives from being able to quantify the odds of a position delta occurring.

The paper’s second goal is to produce a scalable architecture that can process all messages in real-time. The authors state that experiments use four logical cores to host Akka 2.5.1 on Windows 10. Since they did not include any benchmarks, it is not possible to assess their architectural efficiencies. Akka.io (2021) claims to support up to 50 million messages per second. While the underlying platform is efficient, that does not inherently mean the custom application on top retains those characteristics. For instance, the singleton master and consensus nodes might introduce scalability and performance bottlenecks. Another performance concern could stem from too many workers transacting with a shared resource (e.g., cache).

## Expanding Scope

The authors could discuss the technical implementation in greater detail. For example, they discuss using Support Vector Machines (SVM) but leave out the selected feature list. There is also minimal discussion regarding the value-add of Akka specifically. They briefly mention Apache Flink as an alternative but omit any product limitations that led them to Akka.

Second, the publication discusses a distributed architecture that runs on a single machine. Single instanced systems avoid many real-world problems because the network is instantaneous and unlimited. In contrast, multi-node clusters have non-zero transport costs, bandwidth constraints, and processing delays (van Steen & Tanenbaum, 2019). Deferring the discovery of these environmental limitations might miss quality assurance issues or increase production costs.

Third, the paper contains a narrow focus on malicious actors within the AIS information. Other research could expand upon these ideas for sensor networks. Imagine a futuristic factory floor with heterogeneous robots zooming around. Those agents will require scheduling systems that maximize their speed while minimizing accidents. If the scheduler receives some form of Global Positioning System (GPS) data, it will need to support interference potentially. That problem is similar to the spoofed location challenge and requires a mechanism to filter erroneous values.

## Reproducing the experiment

The author’s study relies on Marine Traffic’s open database. It would be possible to purchase the same month’s historical data and rebuild their algorithm. However, afterward, the precise measurements are unlikely to align. These limitations stem from incomplete listings of feature maps, statistical limits, and application source code. The original researchers also mention the removal of extreme outliers. Without more insights into those filter criteria, the reproduction is training on a different data subset. It is also conceivable that the filter removes spoofed locations when finding them is a core objective.

Within these bounds and limitations, there are few unknown details regarding streaming ASI messages through a distributed eventing architecture. Each message comes as a 64-byte structure that is interpretable by any modern programming language. Afterward, several distributed queuing platforms can easily support this design. For instance, funneling the messages into an Amazon Simple Queuing Services (SQS) with Amazon Lambda function would support virtually unlimited scale-out. The results could flow into an Amazon Kinesis stream for a fully serverless and high availability implementation.

## Deriving More Value

Granted, since the architecture tactic is relatively standard, there is less to learn from this perspective. Nearly 80% of data science time goes into curation tasks (Snee, 2015). Instead of stream processing, the researcher could focus on batch processing for faster experimentation. The authors could use something like Apache Spark to drive the workload and confirm their curation decisions. Under a batch processing model, large datasets span across multiple parallel processing units to accelerate computations. Another strength of platforms like Spark comes from the versatility to easily transition back to stream processing after ensuring the batch computations are appropriate.

# References

Kontopoulos, I., Spiliopoulos, G., Zissis, D., Chatzikokolakis, K., & Artikis, A. (2018). Countering real-time stream poisoning. *Big Data Intelligence and Computing and Cyber Science and Technology Congress* (pp. 981-986). Athens, Greece: IEEE. doi:10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00139

Lightbend. (2021, April 5). *Akka*. Retrieved from Akka.io: https://akka.io

Mirakhorli, M. (2014). Identifying and protecting architecturally significant code. *Saturn Talks.* Retrieved from https://youtu.be/B26GVs4tM70

Snee, R. (2015). A practical approach to data mining. *Quality Engineering, 27*, 477-487. doi:10.1080/08982112.2015.1065322

van Steen, M., & Tanenbaum, A. (2019). A brief introduction to distributed systems. *Computing, 98*(10), 967-1009. doi:10.1007/s00607-016-0508-7