Week 2: Domain-Specific Languages

Nate Bachmeier

TIM-7241:Constructive Research Design

May 30, 2021

Northcentral University

# Domain-Specific Languages

Domain-Specific Languages (DSL) enable business professions to describe their rules and policies succinctly. This capability reduces the barrier to automation, promotes cross-disciplinary collaboration, and modulizes complex systems. For instance, financial trading applications contain an execution engine and various investment strategies. These different components require unique professional expertise, making it challenging to have one person build both. Instead, the engineering team can standardize the problem’s shared vocabulary into external files that drive the trading engine. Afterward, domain experts can contribute to those files (scripts) and audit the accuracy of the trading algorithm.

## Recent Use-Cases

Countless scenarios can leverage DSL to automate specific business domains. For example, Cacciagrano & Culmone (2020) built IRON to simplify programming embedded systems through an Event-Condition-Action (ECA) model. The asynchronous nature of embedded systems is challenging to describe in a generic language (e.g., Java). Also, the operators are generally lightweight functions that act upon non-standardized messages. They could mitigate those issues using JavaScript. However, distributed JavaScript functions are difficult to troubleshoot and are not verifiable at build time.

Vernotte et al. (2021) propose the False Data Injection Attach (FDIA) language for identifying security defects within Air Traffic Control (ATC) systems. Building test cases for ATC is incredibly tedious, necessitating an asynchronous programming model with planes moving in multi-dimensional space (3-D + time + control flags). These requirements cause a typical test case to have over 235 parameters. Instead, FDIA supports short scripts that describe the test scenario. Next, it compiles into Automatic Dependent Surveillance Broadcasting (ADS-B) protocol, a standard within ATC validation tools.

## Language Design Challenges

General-purpose languages have extensive add-on modules and troubleshooting tools. However, those same luxuries do not exist within many DSL implementations. Cacciagrano & Culmone (2020) do not address these issues and constrain the developer to a ridge set of transforms. They could extend their language’s schema to allow their interpreter (LUA) to support ANSI C bindings. Vernotte et al. (2021) dialect compiles into a binary replay file and does not need similar extensibility. Since FDIA produces industry-standard files, its users can leverage existing tooling.

## Measuring Effectiveness

Vernotte et al. (2021) want to understand risks to the ATC systems from airplanes reporting malicious and inaccurate information. FDIA scripting reduced the complexity to automate these attack vectors enabling 11,196 unique test cases. This extensive protocol coverage uncovered several critical issues within the ATC.

Cacciagrano & Culmone (2020) focus more heavily on the variability of their EAC design pattern. They use a combination of set and graph theorems to prove these capabilities exist. Their implementation is more theoretical and is missing a real-world deployment (e.g., a Smart Factory or Medical Facility). Without a concrete business case, it is challenging to confirm that the language can handle everyday needs.

## Potential Extension

IRON focuses explicitly on event-driven embedded systems. For instance, a motion sensor triggers a programmable light or camera. However, the EAC model applies to more use-case. Many cloud-native workloads follow a similar EAC pattern using Function as a Service (FaaS) constructs. Instead of interpreting the scripts in-process, the system could generate Infrastructure as Code (IaC) and provision the distributed environment. While creating the code, IRON can continue validating the call graphs.

Kontopoulos et al. (2018) state that maritime surveillance uses an insecure protocol for tracking boat positions. Malicious actors can inject false signals to redirect legitimate traffic and avoid patrols. Their solution relies on an extensive machine learning model. Alternatively, extending FDIA to support naval codes might reduce the overall system’s complexity. Likely this requires significant changes to the FDIA internal code. That would limit the reusability of Vernotte et al.’s (2021) implementation aside from the core idea.

## Measuring Extensions

A GAN (Generative Adversal Network) consists of a generator and discriminator machine learning models. Both components operate within a feedback loop, enabling each side to train the other. Kontopoulos et al. (2018) propose a statistical model for determining that a given ship is malicious. Their solution can serve as the discriminator for generated FDIA navel scripts. The extension’s effectiveness would then equal an F-measurement of this GAN.

Measuring the effectiveness of IRON in the cloud is not nearly as direct. One approach is to assess the amount of engineering time necessary to build a real-world solution. That example project should include Extract Transform and Load (ETL) like operations to demonstrate the usefulness of the language. However, this solution is putting the carriage before the horse. Ideally, constructive design begins with a specific business problem then works backward to a set of mechanisms. This methodology ensures that the correct problems are in focus.

Table 1: Summary of Reviewed Papers

|  |  |  |
| --- | --- | --- |
|  | IRON (2020) | Air Traffic (2021) |
| Authors | Cacciagrano & Culmone | Vernotte et al. |
| Business Problem | Integrating event-driven embedded system | Validating air traffic control protocols |
| Why DSL | Need abstraction layer for heterogeneous topologies | Test cases have 235 parameters on average (e.g., 3-D + time) |
| Challenges | No standard integrations  No third-party library support | N/A – well throughout use-case and aligns with existing standard |
| Artifacts | * LUA Interpreter * Verifiable call graph * ECA design pattern | * Compiles to AIS/ADS-B * Adapter for FDI-Test Framework |
| Measuring Effectiveness | * Abstract analysis (set theory) | * 11196 unique test cases |
| Language Extensions | * ECA is standard within cloud computing | * Naval control protocols |
| Measuring Extensions | * Usefulness wrt, e.g., FaaS | * See: Kontopoulos (2018) * TIM 7140 / W. 6 |

# References

Cacciagrano, D., & Culmone, R. (2020). IRON: Reliable domain-specific language for programming IoT devices. *Internet of Things, 9*, 1-11. doi:10.1016/j.iot.2018.09.006

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

Vernotte, A., Cretin, A., L. B., & Peureux, F. (2021). A domain-specific language to design false data injection tests for air traffic control systems. *International Journal on Software Tools for Technology Transfer, 1*, 1-45. doi:10.1007/s10009-021-00604-4