BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2025
http://excel.fit.vutbr.cz

Static Analysis of Microservices using GraalVM

Vsevolod Pokhvalenko*

Abstract
MicroGraal is a static analysis tool based on GraalVM that extracts communication patterns from Java-based microservice systems. This work extends its capabilities to support WebSocket and GraphQL protocols, introduces automation for multi-service projects, and enhances the intermediate representation based analysis to cover more patterns. The tool was evaluated on open source systems and successfully identified and linked service communication across synchronous and asynchronous paradigms, including STOMP and schema-based GraphQL interactions. The result is a portable and extensible platform for static architectural analysis of modern microservice applications, offering deep insight into control flow, message types, and service dependencies.

*xpokhv00@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

[Motivation] Microservice-based systems are the de facto standard in modern software architecture, promoting modularity, scalability, and independent deployment Figure 1. However, analyzing such systems manually is challenging due to their distributed nature and communication complexity. Tools that help developers understand how services interact, without executing the system, are invaluable for maintenance, security, and optimization.

[Problem definition] Although many tools focus on runtime observability, there is a lack of static analysis solutions that can uncover communication links—such as calls to REST [1], WebSocket [2], or GraphQL [3] directly from compiled Java bytecode. The problem is further compounded by the diversity of communication styles and framework-specific abstractions (e.g., Spring Boot annotations, STOMP [4], etc.).

[Existing solutions] State-of-the-art tools, such as GraalVM, provide low-level insights into compiled Java applications through their Intermediate Representation (IR). GraalVM's Native Image is a technology that compiles Java applications ahead-of-time into standalone executables, generating an IR that can be statically analyzed. However, these tools lack a high-level interface for analyzing components typically used in microservice development (e.g., endpoints, controllers, entities). MicroGraal [5] is a proof-of-concept tool developed for the Java Platform that enables static extraction of service interactions from compiled applications. Prior versions of MicroGraal only supported REST extraction in a narrow setup, with limited project flexibility and no support for asynchronous protocols or schema-based APIs.

[Our solution] This project significantly extends MicroGraal by adding support for the detection of WebSocket and GraphQL communication patterns, in addition to automation tools to analyze arbitrary Maven and Gradle projects. Using the intermediate representation of GraalVM Native Image and a custom analysis plugin, the system reconstructs call graphs, message endpoints, and inter-service dependencies without execution. Figure 6

[Contributions] Key contributions include: (1) a modular extraction system for REST, WebSocket, and GraphQL; (2) integration with Graal IR to reconstruct rich communication metadata like message types and topics; (3) a flexible preprocessing pipeline with classpath resolution and project scanning; and (4) a redesigned visualization frontend. The result is a scalable and extensible static analysis platform tailored to the needs of modern Java microservices.

2. System Overview

This project proposes an approach for static analysis of communication patterns in Java-based microser-

vices by inspecting Graal Intermediate Representation (IR) [6] during native image compilation, without executing the application.

Instrumentation Layer. A custom plugin `Prophet` is injected into the native image build process, enabling direct IR traversal. The plugin identifies method calls, constants (e.g., URL strings) Figure 2, and control flow constructs that reveal communication patterns across services.

Extractor Modules. Two static analyzers were implemented:

- **WebSocket**: Identifies endpoint registration (e.g., `addHandler()`), handler classes (e.g., `WebSocketHandler`), and message paths including STOMP topics Figure 3.
- **GraphQL**: Parses resolver annotations (e.g., `@QueryMapping`), operation names, and argument structures in both queries and mutations Figure 4.

Return/Message Type Inference. An important contribution of this work is the ability to extract return types and data transfer objects (DTOs) from the IR graph. This allows communication edges to include semantic metadata about transferred payloads, which enhances linking and visualization.

Linking Phase. Detected communications are merged into a global dependency graph. REST and WebSocket calls are matched by URI; GraphQL interactions are linked using operation names and types.

Visualization. A React frontend renders protocol-aware service graphs and bounded context maps. Features such as overlays, filtering, and metadata tooltips make analysis results navigable and visually expressive. Figure 5

3. Results and Evaluation

To validate the system, a diverse set of microservice applications was analyzed, covering REST, WebSocket, and GraphQL communication styles, and built using Maven or Gradle. These projects represented open-source use cases.

WebSocket & STOMP Architectures. Microservice systems employing Spring's WebSocket and STOMP stack were used to evaluate the detection of asynchronous communication. The analysis correctly identified:

- Endpoint URIs (e.g., `/messages`)
- Protocol-specific handler classes
- Message directions and mapped payload types (e.g., `MessageDTO`)

GraphQL Microservices. Schema-driven projects were used to assess the extraction of GraphQL operations. The system accurately reconstructed:

- Resolvers defined via annotations like `@QueryMapping`
- Operation types (query, mutation) and names
- Argument values and return/document structures

REST Baseline. Traditional REST-based communication using synchronous clients was successfully parsed and linked, with endpoints and client calls matched via URI and return type inference.

Visual Analysis. The extracted data was rendered into protocol-aware graphs, bounded context maps, and communication overlays—revealing architectural patterns, coupling, and missing dependencies.

Impact. The system demonstrated broad applicability across more than a dozen projects. It enabled static recovery of inter-service dependencies, improved visibility into communication behavior, and provided a foundation for further architectural analysis.

4. Conclusions

This work extends the MicroGraal to support modern microservice communication beyond REST, including WebSocket and GraphQL. By embedding static analysis into the GraalVM Native Image process, it detects communication patterns at the bytecode level using Graal IR.

New modules support schema-aware detection of bidirectional communication and operations, with enhanced extraction of return and message types across services. This improves the semantic accuracy of communication graphs.

Evaluation on diverse projects confirms its effectiveness for analyzing asynchronous and schema-based systems.

Acknowledgements

I would like to express my sincere gratitude to my thesis supervisor, Ing. David Kozak, for his invaluable guidance, support, and valuable insights throughout the development of this project. His expertise in static analysis and compiler technology significantly contributed to the success of this work.

References

[1] Roy T. Fielding and Julian Reschke. Hypertext transfer protocol (http/1.1): Semantics and content. RFC 7231, 2014. https://datatracker.ietf.org/doc/html/rfc7231.

[2] Ian Fette and Alexey Melnikov. The websocket protocol. RFC 6455, 2011. https://datatracker.ietf.org/doc/html/rfc6455.

[3] Facebook Inc. Graphql specification. https://spec.graphql.org/October2021/, 2021.

[4] Jeff Coumans et al. Stomp protocol specification, version 1.2. https://stomp.github.io/stomp-specification-1.2.html, 2012.

[5] R. Hutcheson, H. Gao, D. Kozak, R. Bonett, and H. Sajnani. Software architecture reconstruction for microservice systems using static analysis via graalvm native image. In *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 12–22. IEEE, 2024.

[6] Oracle Labs. Graal ir: An extensible declarative intermediate representation. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=6688214dab5456c75c99f8171846242e09d4f5e3, 2023. Accessed: 2025-04-21.