# Enhancing Web Applications with Dynamic Code Migration Capabilities

Sebastian Heil[1][0000−0003−2761−9009], Jan-Ingo Haas[1][0000−0003−1112−3893], and Martin Gaedke[1][0000−0002−6729−2912]

Technische Universität Chemnitz, Chemnitz, Germany
{firstname.lastname}@informatik.tu-chemnitz.de

**Abstract.** Dynamic migration of code between client and server of a web application allows to balance the needs of users for smooth and responsive user interactions with the interests of software providers to reduce costs and use resources efficiently. The ability to change the execution location of parts of the application logic at runtime means that depending on client capabilities, network speed and the current load of client and server, the code distribution can be optimized. In this demonstration, we showcase dynamic code migration for a sample e-commerce web application. The demonstrator is designed according to our novel DCM architecture and uses its infrastructure to automate compilation of code fragments and manage the migration at runtime, leveraging standardized Web technologies like WebAssembly and WebSockets. Demo participants will be able to interactively control the distribution of code fragments via a control user interface in the browser and interact with the e-commerce web application which was extended so that execution locations of application logic can be observed life. This demo provides a running prototypical implementation of the DCM architecture and aims at inspiring discussions about new possibilities for the Web platform from the widespread support of WebAssembly in all major browsers.

**Keywords:** Web Infrastructure · Software Architecture · Code Mobility · WebAssembly · WebSockets.

## 1 Introduction

Code mobility in distributed systems has been a research interest for a long time since the concept was coined in the most influential paper of ICSE'97 [1]. For web applications, moving code execution between client and server allows to balance the needs of users for smooth and responsive user interactions with the interests of software providers to reduce costs and use resources efficiently. The ability to change the execution location of parts of the application logic at runtime means that depending on client capabilities, network speed, and the current load of client and server the code distribution can be optimized.

The wide support of WebAssembly[1] provides the means for executing non-JavaScript application logic on the client side. Thus, a potentially uniform pro-

[1] https://www.w3.org/TR/wasm-core-1/

gramming language on client and server establishes a new platform on top of which the vision of runtime code mobility becomes achievable for web applications, mitigating a main impediment of previous code mobility approaches [1].

Earlier Mobile Agents approaches in the web like HTML5 Agents [6] also make use of platform uniformity, based on JavaScript/NodeJS, however. Contemporary liquid computing approaches like Liquid.js [2] and Disclosure [4] make use of recent Web standards but are similarly specific to JavaScript. In contrast, WebAssembly opens the web environment's client side for arbitrary languages. We follow the idea of Mäkitalo et al. [5] to leverage WebAssembly modules as portable containers for code mobility, but unlike e.g. Blazor[2] at runtime.

This paper provides a brief overview of our novel software architecture for enabling the migration of application logic units between client and server side of a web application at runtime and outlines the interactive demonstration of an implementation of DCM in a Go-based e-commerce web application.

## 2 The DCM Architecture & Infrastructure in a Nutshell

This section summarizes the DCM architecture and supporting infrastructure for Web Engineers' adoption, shown in fig. 1. A more detailed description of DCM, the related design challenges and design rationale can be found in [3].

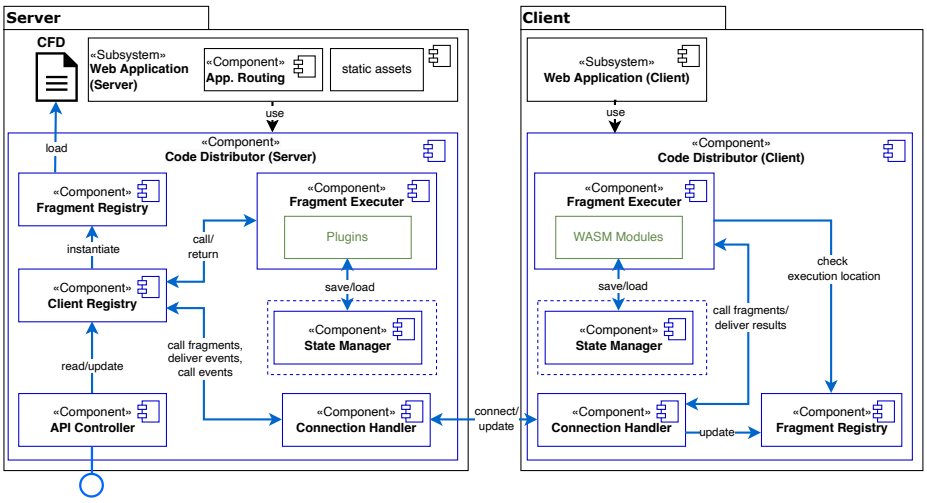


**Fig. 1.** Main Components and Interactions of the DCM Architecture. Supporting Infrastructure is in Blue, Automatically Generated Artifacts in Green.

**Specification and Compilation of Code Fragments.** DCM allows Web Engineers to designate parts of application logic, *Code Fragments*, for migration at

[2] https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor

runtime at the granularity of functions. A fragment is thus defined by its location in the codebase and migration-relevant metadata. Web Engineers specify fragments in a code fragment description (CFD). DCM infrastructure supports CFD creation by automatically extracting metadata such as source code, imports, function parameters/datatypes, through static analysis of the abstract syntax tree. To turn fragments into executable binary artifacts loadable at runtime, they are compiled as plugins for the server side and to WebAssembly modules for the client side. Before compiler invocation, DCM infrastructure modifies the codebase for dependency management and to enable redirection of control/data flow at runtime. The resulting artifacts and DCM runtime environment are then automatically deployed into the web application.

**Dynamic Migration of Code Fragment Execution.** At runtime, client and server `Code Distributor` handle execution of compiled code fragments. They keep track of available fragments and their state and handle fragment life cycle and dynamic migration when requested via the `API Controller`'s REST interface. Client-side fragments are executed as tasks via a pool of WebWorkers. Client/Server communicate via WebSockets to synchronize incoming/outgoing data flows and events. To redirect control & data flow, `Fragment Executer`s serve as proxies between the caller and the fragment code. Corresponding code modifications were automatically made before compilation. Control flow events are passed for fragment invocation/results. Data flow is forwarded and required transformations when entering/leaving the WebAssembly modules and serialization in the WebSocket connection are performed by the DCM infrastructure.

## 3 Demonstration

We built an interactive demonstrator[3] to showcase the DCM capabilities. As shown in fig. 2, the scenario is a sample e-commerce application with basic article list, shopping cart, price/discount calculation functionalities. The application built according to the DCM architecture was extended with UI components to make the migration of fragments at runtime visible and controllable. Whenever application logic from a fragment is executed, an execution location indicator highlights its execution location – server or client. Demo participants can view the list of fragments and control their distribution via a dedicated fragment distribution control, allowing them to change the location of each fragment at runtime. By a second device or participants' own devices, we also demonstrate, that the fragment distribution can be chosen individually for each client, allowing for optimization according to individual load and hardware capabilities. An additional WASM Playground lets participants explore current corner-cases.

## 4 Conclusions & Future Work

DCM enables building web applications that can dynamically adapt execution of application logic at runtime for each client-server pair to better use available

[3] source code available at: https://github.com/heseba/dcm-interactive-demo

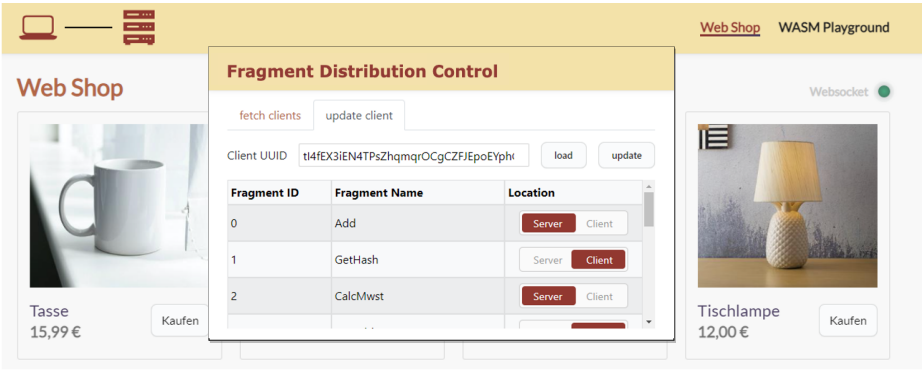


**Fig. 2.** Interactive Demonstrator: DCM-enabled E-Commerce Application with Fragment Distribution Control (center overlay) and Execution Location indicator (top left).

resources and balance users' and software providers' needs. This demo shows a working example using W3C-standardized web technologies and providing interactive code distribution controls. It aims at furthering the discussion about the Web platform in the light of new language-openness through WebAssembly.

Our future work targets the policy-based automation of fragment distribution decisions based on measurements of load and hardware capability detection. A second line of research focuses the combination of DCM with Web Components to enhance capabilities of building micro frontends for dynamic workflows.

**Acknowledgements** The authors would like to thank Alexander Senger for his valuable contributions to the implementation of the DCM demonstrator.

## References

1. Carzaniga, A., Picco, G.P., Vigna, G.: Is Code Still Moving Around? Looking Back at a Decade of Code Mobility. In: Proc. of ICSE'07 Comp. pp. 9–20. IEEE (2007)
2. Gallidabino, A., Pautasso, C.: The LiquidWebWorker API for Horizontal Offloading of Stateless Computations. Journal of Web Engineering **17**(6), 405–448 (nov 2019)
3. Heil, S., Gaedke, M.: DCM: Dynamic Client-Server Code Migration. In: Proc. of ICWE 2023. Springer, Cham (2023)
4. Kim, J.Y., Moon, S.M.: Disclosure: Efficient Instrumentation-Based Web App Migration for Liquid Computing. In: Proc. of ICWE 2022. pp. 132–147. Springer (2022)
5. Mäkitalo, N., Mikkonen, T., Pautasso, C., Bankowski, V., Daubaris, P., Mikkola, R., Beletski, O.: WebAssembly Modules as Lightweight Containers for Liquid IoT Applications. In: Proc. of ICWE2021. pp. 328–336. Springer, Cham (2021)
6. Voutilainen, J.p., Mattila, A.l., Systä, K., Mikkonen, T.: HTML5-based mobile agents for Web-of-Things. Informatica **40**(1), 43–51 (2016)