A Transparent Data Persistence Architecture for the SimJulia Framework

Piet Van Der Paelt
Department of Mathematics
Faculty of Engineering
Royal Military Academy
e-mail: piet.vanderpaelt@mil.be

Ben Lauwens
Department of Mathematics
Faculty of Engineering
Royal Military Academy
e-mail: ben.lauwens@mil.be

Beat Signer
Web & Information Systems Engineering Lab
Department of Computer Science
Vrije Universiteit Brussel
e-mail: beat.signer@vub.be

SimJulia is an open source discrete event simulation framework [1] written in the Julia programming language [2]. The framework transforms processes expressed as functions into resumable functions using the Resumable Functions library [3]. The current SimJulia implementation lacks functionality to persist any variables, characterising the state of such a transformed process during the simulation in a transparent way. To mitigate this shortcoming, we extended both the SimJulia framework and the Resumable Functions library by implementing a transparent probing and persistence architecture, employing language-specific metaprogramming features. Our implementation is based on the Object-Relational Mapping concept (ORM) [4] using the PostgresORM library [5], supported by the PostgreSQL Relational Database Management Systems (RDBMS), and Julia's macro expansion.

A simulation model expressed by means of functions is transformed into resumable functions ready to be run by SimJulia. Our solution uses the code analysis occurring at this stage to store a monitored function's state variables configuration in the database. The user controls which processes are monitored through a flag set on each process. After the code analysis phase, macro expansion takes place, creating the object definitions based upon the configuration saved earlier, before the simulation runs. During the simulation, successive events invoke an array of callback functions being executed. The last function in such an array is the probing function. Monitored processes are probed using the object definitions created just in time. Instantiated objects materialising the state of the concerned process are then persisted in the table specific to the concerned process. The data is subsequently available in the RDMBS, and can be visualised using the technology of choice. Our implementation visualises the data using the VueJS library.

Our contribution consists of a transparent probing and persistence mechanism. Using Julia's metaprogramming capabilities, we were able to divert from

a static ORM configuration as often seen in web applications, to a transparent and dynamic configuration. The end user has the evolution of the state variables available throughout the simulation, without having to provide any configuration.

References

[1] B. Lauwens, "Simjulia: Discrete Event Process Oriented Simulation Framework Written in Julia." `https://simjuliajl.readthedocs.io/en/stable/welcome.html`, 2017. Online; accessed 28-June-2022.

[2] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A Fresh Approach to Numerical Computing," *SIAM review*, vol. 59, no. 1, pp. 65–98, 2017.

[3] B. Lauwens, "ResumableFunctions: C# Sharp Style Generators for Julia," *Journal of Open Source Software*, vol. 2, no. 18, p. 400, 2017.

[4] C. Russell, "Bridging the Object-Relational Divide: ORM Technologies Can Simplify Data Access, But Be Aware Of The Challenges That Come With Introducing This New Layer Of Abstraction," *Queue*, vol. 6, no. 3, pp. 18–28, 2008.

[5] V. Laugier, "PostgresORM.jl: Object Relational Mapping for PostgreSQL." `https://discourse.julialang.org/t/ann-postgresorm-jl-object-relational-mapping-for-postgresql/63847`, 2021. Online; accessed 28-June-2022.