A Data Persistence Architecture for the SimJulia Framework

PIET VAN DER PAELT, Department of Mathematics, Royal Military Academy, Belgium
BEN LAUWENS, Department of Mathematics, Royal Military Academy, Belgium
BEAT SIGNER, Web & Information Systems Engineering Lab, Vrije Universiteit Brussel, Belgium

We present a novel transparent data persistence architecture as an extension of the SimJulia package. We integrated PostgresORM into the ResumableFunctions library by using Julia's metaprogramming support. As such, we were able to remove the dependency on a user's knowledge on architectures for persistence. Our contribution aims to improve the usability, whilst demonstrating the power of macro expansion to move towards a dynamic object-relational mapping configuration.

SimJulia is an open source discrete event process simulation framework (Lauwens et al.). The framework transforms processes expressed as functions into resumable functions using the ResumableFunctions library (Lauwens et al.). Both SimJulia and ResumableFunctions are available through the general registry.

The current SimJulia implementation lacks functionality to transparently store state variables of such a process. The exploitation of the data generated by a SimJulia simulation depends entirely on the user's knowledge on technology for persistence and how to interact with it from within the simulation model. To mitigate this shortcoming, we extended both the SimJulia and the ResumableFunctions packages by implementing a transparent probing and data persistence architecture, employing Julia's metaprogramming support. Our implementation is based on the Object-Relational Mapping (ORM) concept (Russell et al.) using the PostgresORM package (Laugier et al.), supported by the PostgreSQL Relational Database Management Systems (RDBMS), and Julia's macro expansion.

A simulation model expressed using functions is transformed into resumable functions ready to be run by SimJulia. Our solution exploits the code analysis occurring during a first phase of running a simulation to store a monitored function's state variables configuration in the database. This is achieved by using a static ORM configuration which maps an argument of such a function to a tuple in the configuration table.

At the beginning of the second phase, in which the simulation runs, macro expansion takes place, creating the object definitions based on the configuration saved earlier. During this macro expansion, the static ORM configuration is used to retrieve the tuples that define an object within the dynamic ORM. The dynamic ORM provides an object definition for each process within the simulation model, offering the possibility to describe the state of a process through a materialised object. This aspect aside, the dynamic mapping configuration between object and table is also provided through macro expansion.

During the simulation, the standard working of the SimJulia package is to invoke an array of callback functions to be executed for successive events. The last function in such an array is the newly added probing function. When calling the latter, the process under consideration is matched against the object definitions created earlier through

Authors' addresses: Piet Van Der Paelt, piet.vanderpaelt@mil.be, Department of Mathematics, Royal Military Academy, Brussels, Belgium; Ben Lauwens, ben.lauwens@mil.be, Department of Mathematics, Royal Military Academy, Brussels, Belgium; Beat Signer, bsigner@vub.be, Web & Information Systems Engineering Lab, Vrije Universiteit Brussel, Brussels, Belgium.

JuliaCon 2023, July 25–29, 2023, Cambridge, USA

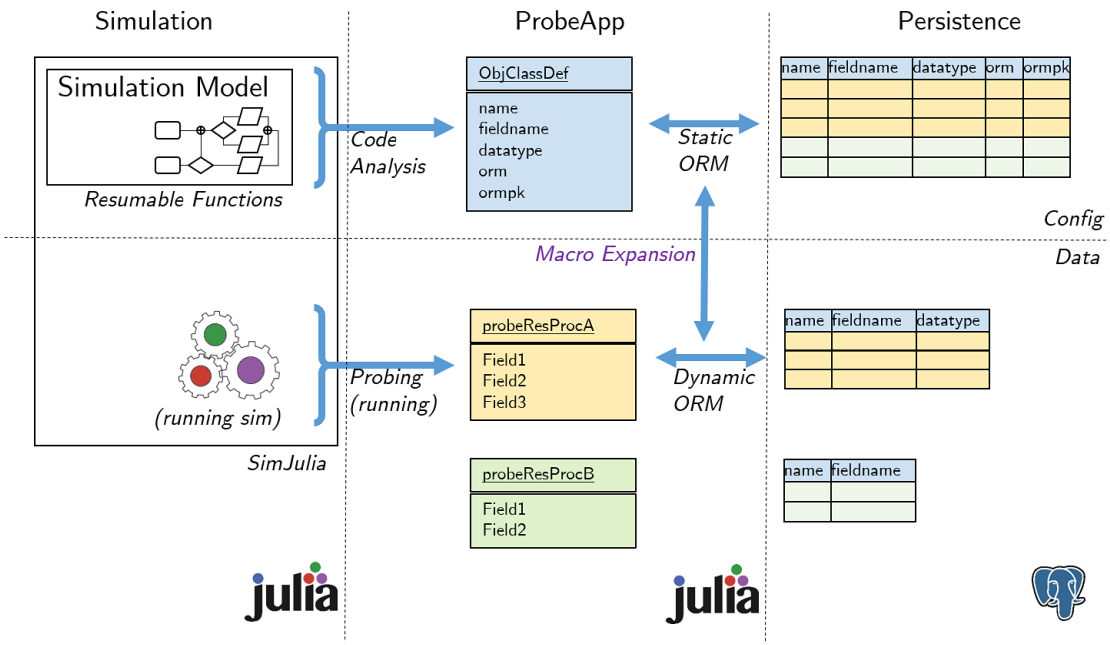

Fig. 1. Data persistence architecture for SimJulia

the dynamic ORM. An object describing the state of the process gets instantiated and is persisted again using the dynamically created ORM.

Configuration is kept in a versioned way in the database, enabling alteration of the simulation model while retaining the configuration of previous versions. The data and object definitions related to any version of the model are subsequently available in the RDMBS, and can be visualised using the technology of choice.

The current proof-of-concept implementation demonstrates the usage of the stored data and the versioned object definitions created dynamically during the simulations using the VueJS package. Through the intermediary of an abstraction layer which employs the dynamic ORM, we were able to provide VueJS with the required view on the data. We are currently considering the possibility to externalise the data based on a REST API for the presented architecture.

Our contribution is twofold. For end users of the SimJulia package, we provide a transparent probing and persistence mechanism, removing the requirement to provide any configuration on this aspect. To the Julia community we demonstrate the usage of metaprogramming to divert from a static ORM configuration as often seen in web applications, to a transparent and dynamic configuration.