Description of Work: Extract Choreography Automata

Student: Gabriele Genovese
Supervisor: Cinzia Di Giusto

February 10, 2025

Abstract

The recent development of concurrent and distributed applications has raised new interest in programming paradigms incorporating threads and message passing into their logic. The use of choreographies can ensure some of the typical properties of concurrent systems (such as liveness, lock freedom and deadlock freedom). ChorEr is a preliminary static analysis tool for a fragment of Erlang programs that generates choreography automata. We plan to build on the existing tool to implement new functionalities and cover more primitives, thus extending the expressiveness of the language under consideration.

Contents

1 General Project Description

Aim. The primary aim of this study is to explore and evaluate various models and paradigms that facilitate the development of robust and scalable concurrent applications. We focus particularly on examining their theoretical foundations.

One of the models used today for concurrent programming in the industrial context is the *actor model*. "Actors", which are lightweight concurrent processes, do not share memory; instead, they communicate by message passing. This model is implemented in a simple way using the Erlang programming language [7], which provides primitives to create new processes (via the `spawn` function) and to **send** and **receive** messages (respectively the `!` and `receive` keywords). Erlang is primarily used to build distributed, scalable and robust applications or to create services that manage thousands or millions of connections and messages daily, such as WhatsApp [18] and Discord [6].

1.1 Motivations

The primary motivation behind this research project is to explore the choreography and choreography automata framework, making it more aligned with the needs of developers, through tools that enable choreography extraction. The tool not only facilitates the visualization and understanding of concurrent systems but also serves as a practical implementation that demonstrates how the abstract concepts of choreographies can be applied to existing technologies. This approach can provide several advantages:

- Debugging: developers can use the tool to gain insights into the global and local behaviors of their concurrent programs;
- Verification of concurrency models: it offers a mechanism to verify that the program's implementation aligns with the intended choreography, ensuring correctness and reducing potential synchronization errors.

1.2 Choreographies

The use of the actor model to conceptualize and create concurrent programs has led the scientific community to focus on solving long-known but challenging behavioral problems and on verifying semantic properties (e.g. liveness, lock and deadlock freedom). One of the mathematical models developed in recent years is **choreographies**: a formal model used to represent systems of communicating processes, enabling semantic proofs regarding the presence or absence of the mentioned properties. Choreographies are **global views** of the behavior of a system, giving a comprehensive perspective of the communication exchanges among actors (also called participants). From the global view, via a simple projection, one can obtain the **local view**, i.e., the individual behavior of each participant in the communication process. Notice that, the local view is limited and actors are unaware of the behavior of the rest of the system.

A *visual* way to formalize choreographies is through **Choreographic Automata** [1], which describe a communication system using a set of finite-state automata. Representing choreographies using FSA is particularly well-suited for illustrating the program flow, as it clearly shows loops and branching, while using previously described results and notions. [17].

The goal of this project is to build a tool that starting from an existing Erlang program extracts the choreographic specification in the form of Choreographic Automata. This amounts to first extract the local views corresponding to each actor and then combine the local views into the global choreography. This step is a sort of inverse operation with respect to the projection and will not be always possible. To illustrate what it means to extract a choreography from an Erlang program, consider the following example.

Example 1.1 (Extraction example)**.** Take two actors: a `main` and a `dummy` process. These two actors subsequently exchange a message. However, one of the actors will execute *send* first and then *receive*, while the other actor will perform the operations in reverse order. Therefore, the expected graph should show a single line of execution, where the exchange of `ciao` occurs first, followed by the exchange of `bello`.

```erlang
-module(simple).
-export([main/0, dummy/0]).

main() ->
    D = spawn(simple, dummy, [self()]),
    D ! ciao,
    receive
        bello -> done
    end.

dummy(M) ->
    receive
        ciao -> done
    end,
    M ! bello.
```

Listing 1: Two processes exchanging messages synchronously

Listing 1 shows to the described scenario in Erlang. The `dummy` process sends the atom `ciao` to the `main` process. This action unlocks the sending of the atom `bello` to `dummy`. Figures 1 and 2 depict the local views of the actors.

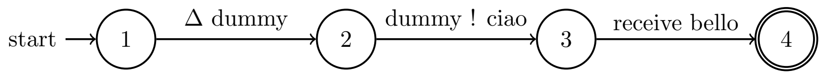

Figure 1: Local view of `main`

After associating the local views with the actors, the algorithm generates the global view shown in Figure 3. As expected, the automaton in Figure 3

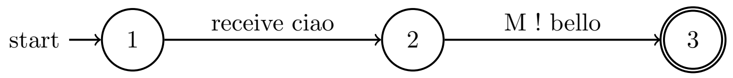

Figure 2: Local view of `dummy`

expresses only one possible global execution of the program: from State 1 to State 2 the actor with the unique identifier `main` spawns the actor with the identifier `dummy`, and then from State 2 to State 3 `main` sends successfully the message `ciao` to `dummy`. Finally, from State 3 to State 4, `dummy` will response to `main` with `bello`.

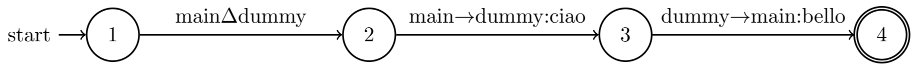

Figure 3: Global view of Code 1

1.3 Framework and challenges

The project centers around ChorEr, a Proof of Concept for a static analyzer developed as part of a Bachelor's thesis at the University of Bologna [9]. This tool is implemented in Erlang and is openly available under the GPLv3 license on GitHub [8]. ChorEr generates multiple DOT files (a commonly employed graph description language) representing Choreography Automata of a given Erlang program's local and global views.

Starting with a Proof of Concept, the greatest challenge will be to formalize and enhance the existing tool effectively. This involves not only refining its core functionalities but also ensuring that the tool will be based on a scientific and well-known theory.

The goal of this project is to focus on improving the accuracy of the tool's static analysis, extending the tool to support a wider range of instructions, and integrating a method for automatic verification of semantic properties.

2 State of the Art

In the industrial context, Erlang's actor model has also inspired other programming languages or frameworks, such as Elixir [19], a programming language based on Erlang's virtual machine. Other programming languages also implement the actor model, such as Go [11] with its GoRoutines (comparable to Erlang's spawns) and channels (similar to Erlang's send and receive).

In the academic context, research in the field of *choreography* focuses on two main topics: *choreography specification* and *choreographic programming.*

- *Choreography specifications*: this area includes formal methods, such as multiparty asynchronous session types [12], which have been established to describe the interactive structure of a fixed number of actors from a global perspective. These methods enable the syntactic verification of actors' correctness by projecting the global specification onto individual participants. Choreography specifications are also studied as contracts, which provide abstract descriptions of program behavior, known as *multiparty contracts* [13].
- *Choreographic programming*: this programming paradigm has been explored both theoretically, as in [2], and industrially, as in [14]. Several choreographic programming languages have been designed and studied to support this paradigm [15, 16, 10, 5].

A way to formalize choreographies is through Choreographic Automata [1], which describe a communication system using a finite-state automaton. By using this model, it is possible to use the results of automaton studies to demonstrate the mentioned properties [17]. Most formal works on communication protocol specifications emphasize the projection of a global specification onto local specifications. However, the process of choreography extraction remains challenging and has been explored in [3], with a general framework for extracting choreographies presented in [4]. In our work, we aim at extracting choreographies from an existing and widely-used programming language like Erlang, despite it not being originally designed with choreography in mind.

3 Current status and workplan

The tool can accurately parse the main constructs and features of the Erlang language. In Table 1, we outline the current status: what is supported, what is not supported, and what are the next objectives. It was decided not to support certain constructs because they are not useful at this stage of the project. For example, supporting error handling with try-catch is unnecessary when variable passing is not yet implemented. Additionally, some constructs were excluded because they are orthogonal to others, such as complex data structures. The tool generates a local view that closely aligns with the behavior of the original actor, and the algorithm for combining local views into a global view performs well on simple examples. However, at present, it does not provide automatic error detection for certain properties in the global choreography.

To address these limitations, we've organized the work plan, as shown in Diagram 3. The primary task is to enhance the tool by adding new features, enabling it to parse and understand a broader set of Erlang programs. Currently, the tool supports only simple functions, therefore there's the urgent need to improve support for functions, recursion, and parameter passing.

The second most critical task is to formalize the data structures and operations within the local view module. This step will improve the tool's internal design, enhance understanding of its functionality, and support the writing of a formal final report, with some proofs. Throughout the development period, we aim to consistently test examples to identify and fix new bugs. Lastly, a benchmarking phase will evaluate the tool's performance and effectiveness.

Keyword	**Support**
atom	yes
integer	yes
float	yes
boolean	yes
tuple	yes
list	yes
record	no
map	no
binary	no
if	yes
case	yes
receive	yes

Keyword	**Support**
!	yes
assignment	yes
function	goal
recursion	goal
hof	goal
when	no
self	yes
spawn	yes
rand:uniform	yes
try catch	no
after	no
math operation	no

Table 1: Supported constructs: Keywords in grey are already supported, keywords in red are not supported and are unlikely to be supported soon. Constructs in green are related to functions, because parameter passing is not supported.

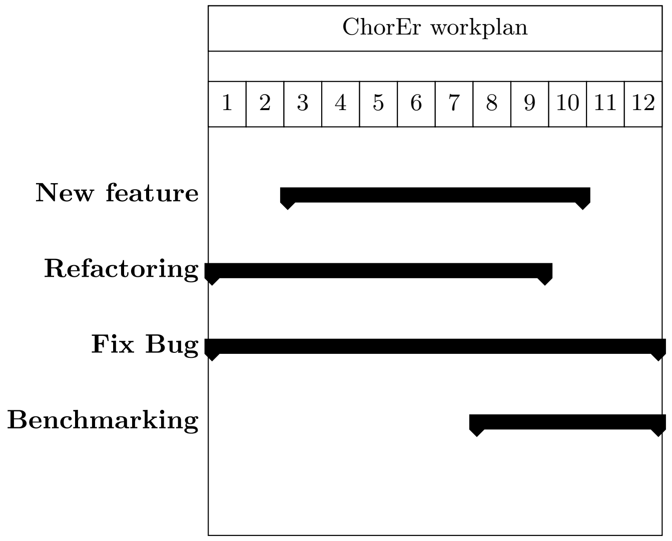

Table 2: Workplan for the project

References

[1] Franco Barbanera, Ivan Lanese, and Emilio Tuosto. Choreography automata. In *Coordination Models and Languages: 22nd IFIP WG 6.1 International Conference, COORDINATION 2020, Held as Part of the 15th International Federated Conference on Distributed Computing Techniques, DisCoTec 2020, Valletta, Malta, June 15–19, 2020, Proceedings 22*, pages 86–106. Springer, 2020.

[2] World Wide Web Consortium. Web Services Choreography Description Language Version 1.0. `https://www.w3.org/TR/ws-cdl-10/`. [Online; accessed 06-June-2023].

[3] Luís Cruz-Filipe, Kim S Larsen, and Fabrizio Montesi. The paths to choreography extraction. In *International Conference on Foundations of Software Science and Computation Structures*, pages 424–440. Springer, 2017.

[4] Luís Cruz-Filipe, Kim S Larsen, Fabrizio Montesi, and Larisa Safina. Implementing choreography extraction. *arXiv preprint arXiv:2205.02636*, 2022.

[5] Mila Dalla Preda, Saverio Giallorenzo, Ivan Lanese, Jacopo Mauro, and Maurizio Gabbrielli. Aiocj: A choreographic framework for safe adaptive distributed applications. In *Software Language Engineering: 7th International Conference, SLE 2014, Västerås, Sweden, September 15-16, 2014. Proceedings 7*, pages 161–170. Springer, 2014.

[6] Discord. How discord scaled elixir to 5,000,000 concurrent users. `https://discord.com/blog/how-discord-scaled-elixir-to-5-000-000-concurrent-users`, 2017. [Online; accessed 18-December-2024].

[7] Ericsson. Erlang/OTP. `https://www.erlang.org/`, 1986. [Online; accessed 26-May-2023].

[8] Gabriele Genovese. ChorEr. `https://github.com/gabrielegenovese/chorer`. [Online; accessed 05-July-2023].

[9] Gabriele Genovese. Chorer: un analizzatore statico per generare automi coreografici da codice sorgente erlang. Bachelor's thesis, University of Bologna, 2023.

[10] Saverio Giallorenzo, Fabrizio Montesi, and Marco Peressotti. Object-oriented choreographic programming. *arXiv preprint arXiv:2005.09520*, 2020.

[11] Google. GoLang. `https://go.dev/`, 2009. [Online; accessed 26-May-2023].

[12] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *Proceedings of the 35th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 273–284, 2008.

[13] Hans Hüttel, Ivan Lanese, Vasco T Vasconcelos, Luís Caires, Marco Carbone, Pierre-Malo Deniélou, Dimitris Mostrous, Luca Padovani, António Ravara, Emilio Tuosto, et al. Foundations of session types and behavioural contracts. *ACM Computing Surveys (CSUR)*, 49(1):1–36, 2016.

[14] IBM. Creating BPMN choreography diagrams. `https://www.ibm.com/docs/en/rational-soft-arch/9.7.0?topic=diagrams-creating-bpmn-choreography`. [Online; accessed 18-June-2023].

[15] Fabrizio Montesi. Jolie: a service-oriented programming language. Master's thesis, University of Bologna, 2010.

[16] Fabrizio Montesi. *Choreographic programming.* IT-Universitetet i København, 2014.

[17] Simone Orlando, Vairo Di Pasquale, Franco Barbanera, Ivan Lanese, and Emilio Tuosto. Corinne, a tool for choreography automata. In *Formal Aspects of Component Software: 17th International Conference, FACS 2021, Virtual Event, October 28–29, 2021, Proceedings 17*, pages 82–92. Springer, 2021.

[18] Erlang Solutions. 20 years of open source erlang: Openerlang interview with anton lavrik from whatsapp. `https://www.erlang-solutions.com/blog/20-years-of-open-source-erlang-openerlang-interview-with-anton-lavrik-from-whatsapp/`, 2018. [Online; accessed 18-December-2024].

[19] The Elixir Team. Elixir. `https://elixir-lang.org/`, 2012. [Online; accessed 26-May-2023].