Polyglot Modal Models through Lingua Franca

Alexander Schulz-Rosengarten*, Reinhard von Hanxleden*, Marten Lohstroh†, Soroush Bateni‡, Edward A. Lee†
* *Kiel University*, Kiel, Germany, {als, rvh}@informatik.uni-kiel.de
† *UC Berkeley*, California, USA, {marten, eal}@berkeley.edu
‡ *UT Dallas*, Texas, USA, soroush@utdallas.edu

***Abstract*—Complex software systems often feature distinct modes of operation, each designed to handle a particular scenario that may require the system to respond in a certain way. Breaking down system behavior into mutually exclusive modes and discrete transitions between modes is a commonly used strategy to reduce implementation complexity and promote code readability.**

The work in this paper aims to bring the advantages of working with modal models to mainstream programming languages, by following the polyglot coordination approach of Lingua Franca (LF), in which verbatim target code (e. g., C, C++, Python, Typescript, or Rust) is encapsulated in composable reactive components called reactors. Reactors can form a dataflow network, are triggered by timed as well as sporadic events, execute concurrently, and can be distributed across nodes on a network. With modal models in LF, we introduce a lean extension to the concept of reactors that enables the coordination of reactive tasks based on modes of operation.

***Index Terms*—coodination, polyglot, modal models, state machines, model-driven engineering, reactors, Lingua Franca**

I. Introduction

The focus of this paper is on reactive systems, which continuously react to their environment, are typically embedded in larger systems, and often have some real-time requirements. Two major notations or views have emerged for describing reactive systems. The *dataflow view* breaks down the program into smaller blocks or actors with streams of data flowing between them. In a *state-oriented view*, the program is modeled in terms of states of the system and its progression in the form of transitions between them. While state machines often describe fine-grained steps at the system level, they can also be used to represent more abstract *modes* of operation [2], where each mode may encapsulate a complex collection of (stateful) reactive behaviors.

However, the languages that provide the capabilities to model systems in any of these notations often come in the form of standalone domain specific languages (as in Simulink, LabVIEW) or language-specific frameworks (such as Akka[1]). The idea of *polyglot coordination* is to allow any mainstream programming languages to benefit from the advantages of modeling with actors, states, or modes. Lingua Franca (LF) [3] embodies this principle, as it is designed as polyglot coordination language based on *reactors*. Reactors encapsulate reactive tasks specified in verbatim code and provide a *minimal coordination layer* around them that is reactive, timed, concurrent, event-based, and accounts for isolated states. The applicability of LF ranges from embedded systems to distributed systems deployed to the Cloud.

[1]https://akka.io/

We here introduce *modal reactors*, a language extension for LF that provides modal models for coordinating polyglot reactive tasks. While this paper is only a brief summary of our work, a more detailed report is available in a separate publication [4].

II. Motivating Example: The Furuta Pendulum

A Furuta pendulum [1] is a classic control system problem. The setup consists of a vertical shaft driven by motor, a fixed arm extending out at 90 degrees from the top of the shaft, and a pendulum at the end of the arm. The goal is to rotate the shaft to impart enough energy to the pendulum that it swings up, to then catch the pendulum and balance it so that the pendulum remains above the arm. Each of these steps (`SwingUp`, `Catch`, and `Stabilize`) requires a different control behavior which makes a controller a prime candidate for a modal model.

From a classical event-driven or dataflow perspective, there is only a single reactive task, computing the motor control based on the angle measurements at the arm and shaft. However, with modes we can identify more fine-granular tasks and coordinate these by embedding them in a modal model. Fig. 1 illustrates our concept of a modal pendulum controller implemented in LF. Our language extension includes the diagram synthesis capabilities of LF, which yields an automatically generated and interactive pictorial representation (Fig. 1a) of the textual program. The program consists of three connected reactors `Sensor`, `Controller`, and `Actuator`. Fig. 1b represents an abbreviated version of the textual source code for the `Controller` reactor. The source code of a more comprehensive implementation is available online.[2]

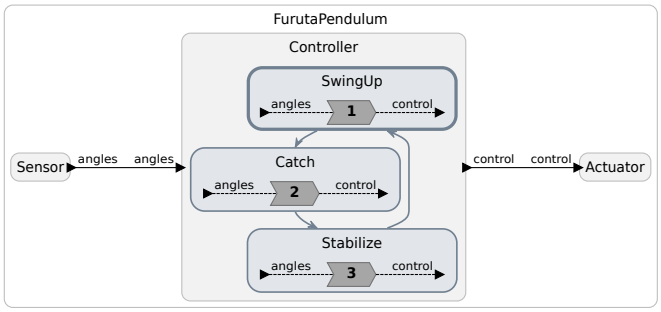
The very first line in Fig. 1b identifies the target language as C, which means that reactors in this file will have their reactions written in C. The first two lines in the `Controller` reactor define the input and output ports. Following are three *reaction* definitions, each reacting to the `angles` input, producing a `control` output, and implementing one of three control laws (abbreviated in lines 7, 14, and 21). We here use the new mode extension to encapsulate each one in a separate mode. Lines 9, 16, and 23 use C expressions (abstracted here) to determine whether a mode change is required, and, if so, invoke `lf_set_mode` to specify the next mode.

III. Modal Reactors

The basic idea of modal reactors is to use the existing reactor model but to allow for a modal coordination, by partitioning

[2]https://github.com/lf-lang/examples-lingua-franca/tree/date23/C/src/modal_models/FurutaPendulum

978-3-9819263-7-8/DATE23/© 2023 EDAA



(a) Structural overview as graphical diagram

```
1  target C;
2  reactor Controller {
3    input angles:float[];
4    output control:float;
5    initial mode SwingUp {
6      reaction(angles) -> control, reset(Catch) {=
7        ... control law here in C ...
8        lf_set(control, ... control value ... );
9        if ( ... condition ... ) { lf_set_mode(Catch); }
10     =}
11   }
12   mode Catch {
13     reaction(angles) -> control, reset(Stabilize) {=
14       ... control law here in C ...
15       lf_set(control, ... control value ... );
16       if ( ... condition ... ) { lf_set_mode(Stabilize); }
17     =}
18   }
19   mode Stabilize {
20     reaction(angles) -> control, reset(SwingUp) {=
21       ... control law here in C ...
22       lf_set(control, ... control value ... );
23       if ( ... condition ... ) { lf_set_mode(SwingUp); }
24     =}
25   }
26 }
```

(b) The `Controller` reactor code

Fig. 1. Aspects of the LF program driving the Furuta Pendulum.

reactors into disjoint subsets that are associated with mutually exclusive *modes*. In a modal reactor, only a single mode can be active at a particular logical time instant, meaning that activity in other modes is automatically suspended. *Transitioning* between modes switches the reactor's behavior and controls the starting point of the entered modes. We support two common options, to reset all elements in a mode or to let them continue based on their previous state. The latter is particularly helpful to control the time-sensitive constructs of LF, such as periodic timers.

While the ideas behind modal reactors are not new, the guidance by LF's fundamental principles towards a polyglot modal coordination layer is novel. Our proposed concept aims at providing a modal extension that is **lean**—a minimal coordination layer that provides the most essential functionality but still offers maximal versatility; **polyglot**—a flexible multi-language wrapper that focuses on the user's language and requires only minor adaptation effort; **concurrent**—allowing the design of multiple separate modal units acting independently; **timed**—a reliable and precise way to specify time sensitive modal behavior; and **deterministic**—yielding unambiguous and reproducible output behavior for the same sequence of tagged input events.

Our modal models in LF embody these very principles and embrace the crucial "black box" approach to reactions, which is the key enabler for LF's polyglot approach. As illustrated by Fig. 1 the modes seamlessly integrate into both the textual and graphical syntax of LF. Modes simply encapsulate reactions and other reactor elements. Transitions are specified as additional effects in the causality interface of reactions, see lines 6, 13, and 20. This enables statically retrieving transition information from the model, e. g. for the diagrams or causality analyses, while the actual behavior is still controlled by the target code inside reactions. The semantics of model transitions are carefully aligned with LF and introduce a microstep delay. This orders subsequently activated modes along the super-dense time of LF and prevents causality issues that would occur if modes would be activated multiple times at the same tag.

Furthermore, the time model of LF is extended to feature *mode-local time*, inspired by modal models in Ptolemy [2]. That means while a mode is inactive, the progress of time is suspended locally. How the timing components behave when a mode becomes active depends on the transition type. For example, a timer in an inactive mode will restart with its initial offset when the mode is entered by a *reset*, whereas if entered via a *history* transition, it will continue counting down the time remaining when the mode was left. The suspension of time gives a clear and consistent meaning to the inactivity of modes, provides a comprehensible state for the mode's contents upon entry, and facilitates modularity.

IV. Conclusion and Outlook

Modal reactors enable the coordination of reactive behavior in terms of modes and transitions. While modes are already central to a large family of existing programming and modeling languages, our approach of building modal abstractions into the polyglot coordination language LF has the advantage of being applicable to a range of target languages at once. Our implementation currently provides modal support for the C and Python targets, demonstrating the versatility of our approach. We also successfully used modes in LF to control a robot and specify different modes for driving and collision avoidance.

Having modes modeled explicitly opens up new opportunities for static analyses, which we plan to further explore in the context of model checking and federated LF programs.

References

[1] K. Furuta, M. Yamakita, and S. Kobayashi, "Swing-up control of inverted pendulum using pseudo-state feedback," *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, vol. 206, no. 4, pp. 263–269, 1992.

[2] E. A. Lee and S. Tripakis, "Modal models in Ptolemy," in *3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT)*, vol. 47. Linköping University Electronic Press, Linköping University, 2010, pp. 11–21.

[3] M. Lohstroh, C. Menard, S. Bateni, and E. A. Lee, "Toward a Lingua Franca for deterministic concurrent systems," *ACM Transactions on Embedded Computing Systems (TECS), Special Issue on FDL'19*, vol. 20, no. 4, p. Article 36, May 2021.

[4] A. Schulz-Rosengarten, R. von Hanxleden, M. Lohstroh, S. Bateni, and E. A. Lee, "Modal reactors," January 2023. [Online]. Available: https://arxiv.org/abs/2301.09597