Available online at www.sciencedirect.com

ScienceDirect

Procedia Engineering 201 (2017) 684–689

Procedia Engineering

www.elsevier.com/locate/procedia

3rd International Conference "Information Technology and Nanotechnology", ITNT-2017, 25-27 April 2017, Samara, Russia

The templet parallel computing system: specification, implementation, applications

S.V. Vostokin*

Samara National Research University, 34, Moskovskoye shosse, Samara, 443086, Russia

Abstract

The article describes the implemented prototype of the Templet parallel computing system for the C++ language. The system uses a novel version of the actor execution model. The design of the actor model makes it possible to define the behavior of the parallel program developed in the Templet system mathematically strict with a temporal logic formula. We believe that this feature of the system is critical, because it gives application developers the freedom to implement actors on any desired platform. Our variant of the actor model can be easily reimplemented for different hardware and in different programming languages, at least for multithreaded execution in shared memory. The article defines the Templet actor model in terms of Temporal Logic of Action, discusses the system design, and shows some examples of its practical use.

Keywords: actor model; execution model; temporal logic; parallel computing system; platform independence

1. Introduction

The Templet system is a toolkit for concurrent programming in terms of the actor model. The system has a compact runtime library to implement the actor semantics of execution. In order to make the system practical and easy to use, the domain language (referred as the Templet markup language) is implemented. The preprocessor converts the high-level descriptions of actors into the Templet DSL including a protocol for message exchange, and the rules of interconnections between actors into the code in C++ language. This generated code has a form of a skeleton. It is a framework with extension points where the user can put their C++ code to complete the program. The programming system includes a preprocessor and several runtime libraries that can be found on the project's GitHub page: *https://github.com/templet-language*.

The article has the following structure. First, we define the formal specification of the variant of actor programming model for the system. We use the temporal logic of actions (TLA) introduced by Leslie Lamport [1] for the specification. Then, we describe a runtime library that implements the given specification. After that, the binding between the runtime library and the application code is discussed. Finally, we make a brief overview of some practical examples developed in the Templet computing system.

* Corresponding author.
E-mail address: vostokin_sv@ssau.ru

1877-7058 © 2017 The Authors. Published by Elsevier Ltd.
Peer-review under responsibility of the scientific committee of the 3rd International Conference "Information Technology and Nanotechnology".
10.1016/j.proeng.2017.09.683

2. Motivation

There are many programming systems with actor support that look more or less the same. The purpose of the research is to demonstrate that: (a) a practical actor system may be very simple; (b) the actor system needs no special features or support in implementation language or runtime; (c) the actor model may be thought as a lightweight extension to the standard multithreading, while the multithreading is a common feature for almost any programming system.

In view of the announced advantages, our variant of the actor model implementation also preserves the other favorable characteristics of actors: absence of race conditions, absence of deadlocks, and a view of a parallel system as a composition of sequential systems. These features make the actors more convenient than threads in application programming.

3. Methods

The method that we use to reach the intended purpose of the study are: (a) to provide basic/minimal support of actors; (b) to provide a binding mechanism between actor code and other user code that is not imposed on language features (e.g. it does not use templates or another extended C++ features).

The first part of the method is based on the specification of our variant of actor model with the temporal logic of actions – a general concurrent model devised by Leslie Lamport.

The second part of the method is based on the use of the domain-specific language (DSL). The purpose of the DSL is (a) to describe the details of actors behavior 1 (e.g. types of receiving messages, one-sided communications, request-response communications, complex actor structures, and so on) in the DSL (not in C++) in the beginning; and after that (b) to generate a C++ skeleton from the DSL description.

There are many actor model implementations, which were proposed since the early works of Carl Hewitt and his colleagues [2]. The approach discussed in this article contrasts with the following implementations: (a) traditional libraries (Akka [3], Intel TBB [4]); (b) entirely new programming languages with actor semantics (Scala [5], Go [6], Erlang [7]); (c) extensions to existing languages (ACT++ [8], Axum [9]).

We propose a language-oriented [10] DSL-based system that is a kind of a skeleton programming tool [11] for programming using actors. Therefore, the system is natively compatible with the existing compilers, libraries, and integrated development environments. This greatly simplifies the design and the use of the system. Our system has many similarities with the model-driven development approach [12].

Many studies have been made to adequately formalize the actor model [13]. However, the approach that uses the temporal logic of action (instead of the classical functional approach) for presenting actor semantics with variables and changing of their values is new and promising. This paper attempts to draft a specification only for the variant of the actor model implemented in the Templet system. We do not claim complete formalization of the actor model.

4. Specification of the Templet Actor Model

The Templet system was designed for developing programs with actor execution semantics. Our method of actor model implementation focuses on passing the access rights to objects and the activity between actors rather than on classical message passing. This approach enables us to build a precise specification and a compact implementation of the actor runtime.

Let Var be a set of all program variables. We suppose that there is a function F that defines to which actor or message a variable belongs:

$$F: Var \to \{actor, message\} \times N, \tag{1}$$

where a number from set N denotes an actor or message identifier. In a specific implementation (namely in C++) this relation may be expressed with a declaration of the variable inside an actor class or a message class, or may be simply implied by the programmer.

Let us assume that the set

$$\{a[1], a[2], \cdots, a[i], \cdots b[1], b[2], \cdots, b[j], \cdots, m[1], m[2], \cdots, m[j], \cdots\} \tag{2}$$

denotes all the runtime variables with the following components: (a) the activity of the actor i is expressed as $a[i] \in \{0,1\}$, where Boolean 0 means not active, and Boolean 1 means the actor is active, and it handles a message; (b) the message activity is expressed as $m[j] \in \{0,1\}$, where Boolean 0 means that the message is already received by the actor, and Boolean 1 means that the message is transmitted; (c) the array $b[\cdot]$ of runtime variables stores an identifier of an actor to which message j belongs. A message belongs to an actor, if this message is being transferred to this actor at the moment, or has been already transferred to this actor.

By using these designations we can express the runtime atomic actions that change the system state.