Information and Software Technology 79 (2016) 17–35

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

A Linda-based platform for the parallel execution of out-place model transformations

Loli Burgueño[a,*], Manuel Wimmer[b], Antonio Vallecillo[a]

[a] Universidad de Málaga, Atenea Research Group, Bulevar Louis Pasteur, 35. (29071) Málaga, Spain
[b] Vienna University of Technology, Business Informatics Group, Karlsplatz 13. (1040) Vienna, Austria

ARTICLE INFO

Article history:
Received 1 April 2015
Revised 7 May 2016
Accepted 6 June 2016
Available online 28 June 2016

Keywords:
Model transformation
Performance
Scalability
Parallelization

ABSTRACT

Context: The performance and scalability of model transformations is gaining interest as industry is progressively adopting model-driven techniques and multicore computers are becoming commonplace. However, existing model transformation engines are mostly based on sequential and in-memory execution strategies, and thus their capabilities to transform large models in parallel and distributed environments are limited.

Objective: This paper presents a solution that provides concurrency and distribution to model transformations.

Method: Inspired by the concepts and principles of the Linda coordination language, and the use of data parallelism to achieve parallelization, a novel Java-based execution platform is introduced. It offers a set of core features for the parallel execution of out-place transformations that can be used as a target for high-level transformation language compilers.

Results: Significant gains in performance and scalability of this platform are reported with regard to existing model transformation solutions. These results are demonstrated by running a model transformation test suite, and by its comparison against several state-of-the-art model transformation engines.

Conclusion: Our Linda-based approach to the concurrent execution of model transformations can serve as a platform for their scalable and efficient implementation in parallel and distributed environments.

1. Introduction

Model Driven Engineering (MDE) is an approach to software development where models and model transformations play a central role in all software engineering processes [1]. Models capture the aspects of interest of systems and behave as an abstraction of them representing reality for a given purpose. Thus, models are simpler, safer and/or cheaper than reality and allow users to deal with the interesting parts of the systems in a simplified and more focused way. In turn, model transformations are in charge of manipulating these models. They permit generating system implementations from high-level models, conducting model analysis, software migration and modernization [2] or even data integration [3].

To support such model transformation scenarios, a wide range of different transformation languages already exists, each of them comprising different characteristics [4]. However, the increasing size and complexity of models are challenging the existing model transformations languages and engines, whose performance and scalability need to be significantly improved as the industry is progressively adopting model-driven techniques [5]. In particular, most MDE solutions and tools are having problems for coping with models of only several millions of elements since most environments require the models to reside in memory. Their scalability is not sufficient either, and performance rapidly degrades as the size of the models grows beyond a few million elements. Furthermore, current model transformation engines are mostly based on sequential and in-memory execution strategies and thus they have limited capabilities to transform large models in acceptable time. This hinders the benefits of using models and model transformations in different application domains that use large models.

At the same time, parallel computing has become increasingly important as chipmakers are putting more and more processor cores on individual chips — which are mainly wasted if sequential engines are used. Similarly, distributed algorithms are gaining attention as computer communication is getting much faster, cheaper and more reliable, and the Cloud is taking over.

In this paper we present a framework, called LinTra, to achieve the parallel and distributed execution of transformations, providing significant performance and scalability improvements with regard

* Corresponding author.
E-mail address: loli@lcc.uma.es (L. Burgueño).

http://dx.doi.org/10.1016/j.infsof.2016.06.001
0950-5849/

to existing model transformation solutions. We make use of data parallelism to achieve parallelization, and follow the Linda [6] coordination model that offers concurrency and distribution mechanisms using the well-known principles of separation of concerns [7], permitting concurrent access to distributed data in a transparent way. In LinTra, distribution is achieved using the *blackboard* [8] distributed shared memory approach, which also provides an abstraction over existing Java-based data space platforms. Scalability is addressed by using data management middleware platforms to implement the tuple space, which are able to deal with very large volumes of distributed data in an efficient way. Finally, the *master-slave* pattern [8] is used for achieving data parallelism.

Based on initial ideas outlined in our previous works [9,10,11], the contribution of this paper is fourfold. First, we present a novel Java-based execution platform called LinTra for the parallel execution of out-place transformations that may also be used as a target for high-level transformation language compilers. Second, we provide a mapping of model transformation concepts into the LinTra framework. In particular, we define the representation of models and metamodels and how those models are stored over a set of machines using a blackboard approach. Third, we demonstrate the performance and scalability of this platform by reporting the results of running a model transformation test set using different Java middleware platforms for presenting models, and by comparing it against several state-of-the-art model transformation engines, including sequential and parallel ones. Finally, we discuss some implementation solutions for dealing with models that do not fit in memory or which are distributed over several machines, using distributed, scalable NoSQL databases [12] as underlying technologies.

The structure of this paper is as follows. Section 2 introduces the LinTra framework and how model transformations are embedded in this framework, and then Section 3 focuses on LinTra's features for out-place transformations. LinTra is evaluated by a case study in Section 4, where we investigate the execution performance of LinTra with respect to different Java-based middleware platforms used to store and retrieve models, and we compare LinTra with other execution engines. Finally, in Section 5 we discuss related work and conclude the paper in Section 6 with an outlook on future work.

2. LinTra: preliminaries for marrying Linda with model transformations

2.1. The Linda coordination model

Linda is a mature coordination model that uses a shared memory space as the only means of communication among parallel processes. This model provides a set of coordination primitives that can be added to existing programming languages for parallel and distributed processing. It was first proposed by David Gelernter at Yale University in the mid-1980s [13] and in recent years there has been a resurgence in interest in it, particularly with regard to Java implementations of Linda [14,15].

In distributed memory systems, such as networks of workstations, the shared memory, which is called *tuple space* or *blackboard*, is usually distributed among the processing nodes. Independent from the implementation strategy employed, the tuple space is structured as a bag of tuples. An example of a tuple with four fields is ("circumference", 3, 47, 53), where 3 is the radius, and 47 and 53 indicate the position (x and y coordinates) of the circumference represented by this tuple. Another example is ("square", 5, 20, 30) which represents a square whose side length is 5, whose position on the X-axis is 20 and 30 on the Y-axis.

Linda provides several operations as coordination primitives to place tuples into a tuple space (write operation) and to retrieve tuples from it (read operations). Read operations can be either blocking or non-blocking, and remove the read element or not. A piece of Linda code with examples of these operations is shown in Listing 1.

```
1 write(``circumference", 3, 47, 53)
2 write(``circumference", 7, 30, 21)
3 write(``square", 5, 20, 30)
4 read(?, ?, 20, ?)
```

Listing 1. Example of Linda pseudocode.

The specification of the tuple to be retrieved (line 4) makes use of an associative matching technique whereby a subset of the fields in the tuple have their values specified. In our example, the read operation defines a pattern that matches all the tuples whose position on the X-axis is 20. Therefore, the tuple written in the third line is retrieved.

As a coordination language, the Linda primitives were conceived to be integrated with a programming language, which is called the host language. There are different Linda implementations for different host languages such as C-Linda [16] for C and JavaSpaces [17] for Java.

2.2. LinTra

LinTra is a framework that allows the parallel execution of out-place model transformations, no matter if the models are located in a single machine or distributed over a set of nodes. We base our transformation approach on Linda, implementing a shared tuple space (or blackboard) and using data parallelism [6].

Fig. 1 shows the architecture of LinTra. For running transformations on such an architecture, we explored how model transformations fit into the Linda framework and we made the distinction between two independent layers. The middleware layer contains the concrete Linda implementation, while the LinTra layer on top of it comprises the model transformation written in Java and the models and metamodels representations. We also decided how model transformation trace links are encoded to allow for efficient retrieval, and how the transformation rule execution is distributed over the available computational resources (machines, cores, etc.). Note that, although we implemented our solution using Java, we could have use other languages such as C, C++, C#, etc.

2.3. Linda and existing implementations

There is a wide variety of pure Linda implementations written in different languages such as JavaSpaces [17] and TSpaces [18] in Java, C-Linda [16] in C, Rinda [19] in Ruby and PyLinda [20] in Python.

In addition, there are other mature software solutions for data management based on in-memory data grids (IMDG) or on distributed caches that are not used as Linda implementations but that provide similar functionality and even more. They are a specific kind of NoSQL databases called key-value caches. In particular, they (*i*) scale-out because every node (computer) adds its CPU and RAM to the cluster which can be used by all the nodes; (*ii*) can store big data and enable fast access to it as it is manipulated in main memory; (*iii*) permit dynamic scalability as nodes can dynamically join other nodes in a grid (cluster); (*iv*) enable elastic main memory as every node adds its own RAM memory to the cluster's memory pool; (*v*) implement fault-tolerance mechanisms without data loss, and (*vi*) implement a programming model to access the cluster as if it was a single machine. Some of these data management solutions are Hazelcast, Oracle Coherence, GigaSpaces XAP, Ehcache and Infinispan, to mention a few. In Section 4 we present a brief description for each particular solution we have worked with.