Concurrent and Distributed Model Transformations based on Linda

Loli Burgueño

GISUM/Atenea Research Group, Universidad de Málaga
Bulevar Louis Pasteur, 35 (29071), Málaga, Spain
loli@lcc.uma.es
http://www.lcc.uma.es/~loli

Abstract. Recently Model-Driven Engineering (MDE) is becoming more and more popular as it is able to solve complex problems by exploiting the abstraction power of models. As models, metamodels and model transformations are the heart of MDE, they play a vital role. Nevertheless, existing transformation languages and accompanying tools cannot deal with large models such those used in the fields of astronomy, genetics, etc. The main problems are related to the storage of very large models, the unreasonable time needed to execute the transformation and the impossibility of transforming distributed or streaming models. We tackle this problem by means of incorporating the concurrent and distributed mechanisms that Linda (a mature coordination language for parallel processes) provides into model transformation approaches.

Keywords: MDE, model transformation, concurrency, distribution, Linda

1 Introduction and Problem

MDE is a relatively new paradigm which has grown in popularity in the last decade. So to speak, model transformations (MTs) are, together with models and metamodels, the key of MDE, allowing to systematically manipulate models. MTs can be classified according to different characteristics [6, 19]: abstraction level of input and output models (i.e., horizontal vs. vertical transformations), kind (i.e., model-to-model, text-to-model or model-to-text), directionality (i.e., uni-directional vs. bi-directional transformations), manipulation of input and output models (i.e., in-place vs. out-place transformations), etc. And there are several different intents for which transformations are applied [1]: abstraction, refinement, synthesis, model composition, etc.

Because of this increasing variety of MTs scenarios, there exists a wide range of different languages for developing MTs, each of them comprises different characteristics [5]. Some examples of MTs from different language categories are GrGen (graph transformation) [14], Kermeta (imperative) [20], QVT-R (declarative) [12], ATL (hybrid) [16] and UML-RSDS (general purpose MDE tool) [18].

Lately, the MDE paradigm is being embraced by companies, thus, MTs are extensively used and the problems being addressed are increasingly complex.

The main reason for its use is that there are scenarios where MDE is the most appropriate option to solve their problems or necessities. For instance, generating code or migrating the architecture/software/data is very appropriate for using MTs. However, state-of-the-art tools and languages have several limitations [17] as:

Scalability They do not allow to work with big models with millions elements.
Performance The MTs for some well-known tools take a long time even for medium size models (around 250,000 model elements).
Concurrency They do not support concurrent transformations. Although the machines of nowadays count on several cores, they are not taking advantage of the full IT infrastructure.
Distribution They are not able to deal with models distributed among different machines. Nowadays, and given the trend to move towards *Cloud Computing*, distribution is really needed for ensuring dynamic scalability.
Streaming They need to read the complete model into memory before starting the transformation and do not allow that the model is a data flow.

The research question we address is "*Can the performance of MTs be improved so that MTs are used for industrial practices?*" which leads to the following: (1) *Can MTs be executed in parallel?* and (2) *Can models be distributed and/or stored in the cloud?*

2 Related Work

The scalability problems of loading large models represented by XMI documents into memory have been already recognized several years ago. One of the first solutions for EMF models is the Connected Data Objects (CDO)[1] model repository which allows to store models in all kinds of database back-ends such as traditional relational databases or emerging NoSQL databases. CDO supports the ability to store and access large-sized models due to the transparent loading single objects on demand and caching them. If objects are no longer referenced, they are automatically garbage collected. There are also several emerging projects that are considering to store very large EMF models, like MongoEMF[2] and Morsa [7]. Both approaches are built on top of MongoDB, which is used as storage technology. In [4], Clasen et al. elaborate on strategies for storing models in a distributed manner by horizontal or vertical partitioning into the Cloud.

So far, we are reusing the storage of Gigaspaces Technologies [10] by transforming models and their associated metamodels to a tuple representation to inherit the good scalability from the underlying technology.

Several lines of work consider the transformation of large models. We plan that our work is focused on out-place model transformations, i.e., loading input model and producing the output model from scratch by applying all matching

[1] `http://projects.eclipse.org/projects/modeling.emf.cdo`
[2] `http://code.google.com/a/eclipselabs.org/p/mongo-emf`

transformation rules. However, to deal with large models, orthogonal techniques may be applied. Especially, two scenarios have been discussed in the past that benefit from alternative execution strategies. First, if an output model already exists from a previous transformation run for a given input model, only the changes in the input model are propagated to the output model. Second, if only a part of the output model is needed by a consumer, only this part is produced while other elements are produced just-in-time. For the former scenario, *incremental* transformations [15, 21] have been introduced, while for the latter *lazy* transformations [23] have been proposed. In this work, we propose a fundamental approach for parallelizing model transformation executions that may be also combinable with incremental and lazy transformations. Furthermore, it can be generalized to other types of MTs (e.g., in-place MTs).

Another interesting line of research for executing transformations in parallel is the work on critical pair analysis [13] in the field of graph transformations. This work has been originally targeted to transformation formalisms that do have some freedom for choosing in which order to apply the rules. Rules that are not in an explicit ordering are considered to be executed in parallel if no conflict, e.g., add/forbid or delete/use conflicts, is statically computed. However, the execution engines follow a pseudo-parallel execution by going back to a sequential application of the rules. But the general notion of critical pairs may be also a valid input for distributing transformation rules. In particular, having non-conflicting transformation rules allows for distributing them easier without having negative side-effects.

The performance of model transformations is now considered as an integral research challenge in MDE. For instance, Amstel et al. [24] considered the runtime performance of transformations written in ATL and in QVT. In [26], several implementation variants using ATL, e.g., using either imperative constructs or declarative constructs, of the same transformation scenario have been considered and their different runtime performance has been compared. However, these works only consider the traditional execution engines following a sequential rule application approach. The only work we are aware of dealing with the parallel execution of transformations is [4] where Clasen et al. [4] outlined several research challenges when transforming models in the cloud. In particular, they discussed how to distribute transformations and elaborated on the possibility to use the Map/Reduce paradigm for implementing model transformations.

3 Proposed Solution

Our goal is to provide model transformations with concurrency and distribution, addressing the problems of storing and handling large models, distributed models and models which are available as a infinite stream of elements.

Instead of starting from scratch dealing with the partitioning of models into *model slides* and its physical location in the set of available machines, we propose to make use of Linda [9], a mature coordination language for parallel processes. Linda implements a shared tuple space that can be distributed over a set of

machines and accessed in parallel. All those features are implemented in an user-transparent way and Linda only provides to the user the primitives to read from the tuple space and to write to it.

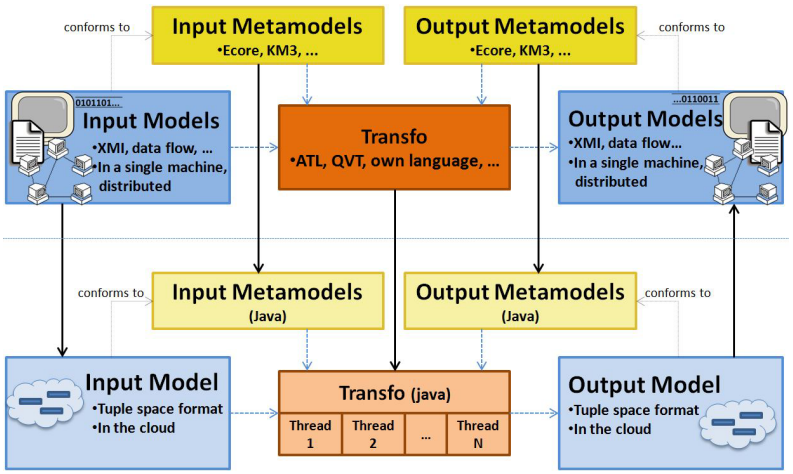

Fig. 1. Architecture for executing MTs based on Linda

Then we plan to create a model transformation process according to the schema that Figure 1 shows. In the figure we can see two layers, one in the top and another in the bottom, separated by a dotted line. To the user's only the top layer is visible and he/she will only need to provide the model transformation, the input and output metamodels involved and the input models. The model transformation will be required either in a well-known language such as ATL or QVT or in a new language that we plan to build. The input models may be given as an XMI file, a set of XMI files, a data flow, several data flows, etc. and can be located in a single machine or can be distributed over a set connected machines.

Internally, the system will transform the model transformation given by the user, which is written in what we call a *high level* model transformation language, to a *low level* transformation language that supports distribution and concurrency and that makes use of Linda. This *low level* language as well as the Linda implementation can be, for example, Java code.

Furthermore, the system will translate the input models to the model representation required by the *low level* transformation and the concrete Linda implementation, i.e., the objects belonging to the models will be transformed into tuples and stored in the input tuple space. This model representation and *low level* model transformation language will be designed taking into account characteristics as, for example, readability and semantic, but most importantly efficiency.

Once there is data available in the input tuple space and the transformation has been transformed to the *low level* transformation language the transformation process will start. For it, several threads will be launched and concurrently each thread will execute the following steps: (1) read objects (model elements) from the input tuple space, which can be distributed or not, (2) transform the objects (create the output objects from the input ones), (3) store the output objects in the output tuple space, (4) go back to 1. if there are more objects in the input tuple space.

4 Preliminary Work and Current Status

The first step in order to implement model transformations over Linda is to count on a Linda implementation. So far, we have used the Java implementation of an in-memory data grid offered by GigaSpaces Technologies [10] called *XAP Elastic Caching Edition*.

Our work is still in an early state but so far we have taken several design decisions. For instance, how to represent models and metamodels as tuples, how many independent tuples spaces are required, how the elements are distributed among those tuples spaces, how the transformation rules are executed in parallel, study the need of encode and store trace links, etc. Some of these points were presented in [2].

First of all, as the Linda implementation is in Java, the models and metamodels are represented in Java too. The metamodels are represented by means of Java classes where each meta-class is represented by a Java class and each attribute or reference is a Java field. The inheritance between classes is also represented by Java inheritance. Regarding the models, they are composed of class instances, i.e. Java objects. Furthermore, in order to support EMF, we are working on a code generator that provides the translation from Ecore-based metamodels to the Java classes.

We have decided that our concurrent and distributed model transformation will be an out-place transformation language and for it we consider two different tuples spaces, one of them containing the input models and another containing the output models. The previously mentioned *low level* transformation language is written in Java too. Nevertheless, we consider to experiment with Scala in addition due to its support for concurrent and functional programming.

As the transformation is executed concurrently, the dependencies among the objects in the models need to be solved. In order to achieve that, we have given to each input object a string attribute with an identifier (ID) which makes it unique. Then, for each relationship between two objects, an attribute with the target object ID is stored in the source object. We also have built a function, F, similar to the resolve methods in current transformation languages, that is able to compute the identifier of the output element given the identifier of the input element. In case of dependencies when transforming an object, counting on *F*, it is not necessary to wait until the dependent object has been transformed

because we can know its ID beforehand and, thus, create the relationship in the output model.

Finally, the last thing to be defined is how to assign the code to execute by each thread and the objects to transform. So far we have applied the *process farm* approach where all the threads execute the same code. Then, every object to transform is filtered and the corresponding rule (piece of code) is executed. The problem is that there exist rules which take longer to finish their execution. Let us suppose the hypothetical situation where our machine has two cores, and we have a transformation with two rules. If all the objects assigned to the first thread need to be executed by the heaviest rule and all the objects assigned to the second thread are executed by the lightest rule, the first core will end very soon in comparison with the second thread, thus, the performance will not be the most efficient. The execution time could improve changing the assignment of objects to the threads. For that we have to work on several points, establishing metrics to measure the weight of the rules, i.e., the time their execution takes and creating an algorithm to do the optimal assignment. For the metrics, we plan to extend the work on the complexity of the OCL expressions [3] with model transformation specific metrics and for finding an optimal assignment to use genetic algorithms such as harmony search.

5 Expected contributions

So far, we have analyzed the existing implementations of Linda and we have adopted the option that best fits our needs. We have also created a metamodel and model representation according to that implementation. Nevertheless, after some experiments, we have realized that, for concrete cases, small variations of this representation might improve the performance. For that, we plan to analyze carefully each scenario and take it into account when translating the models and metamodels given by the user to our representation. Furthermore, we do not discard to change the whole Linda implementation and, thus, the model and metamodels representation in the future if we find some deficiencies that make it an unsatisfactory solution for our requirements or if we find another one more efficient. We neither discard to create our own Linda implementation.

The parallelization of the model transformation execution is in an early stage. As we presented in Section 4, the performance depends on several parameters and we still have to find the optimal configuration for them in order to finish the model transformation execution as quickly as possible.

Until now, we have not worked in the top layer reflected in Figure 1. As next step, we are developing Higher-order Transformations (HOTs) [22] for producing the Linda-based transformations on the lower level from transformations written in ATL and QVT relations. We will probably define our own syntax and create a *high level* language from scratch if we realize that that will improve considerably the performance.

To sum up, we have just reached the first steps but, at the end of this work, we expect to count on a concurrent and distributed model transformation engine.

6 Plan for evaluation and validation

We plan to make a comparison between our approach and the most well-known and extended languages (such as ATL, QVT relations, etc.) by mean of executing the same model transformation over the same set of input models and evaluating the time they take. Nevertheless, when the input models are big enough to be transformed by the existing languages, we will have missed the reference to break. Then, our aim will be to finish the transformation in the shortest possible time. Some preliminary results of our work can be found in [2].

We will need a methodology for assessing the quality of model transformations. With that methodology we will be able to measure the speedup of transformations, the best and the worst execution times and the kind of runtime complexities that current model transformations have, i.e., we will know the complexity class (linear, polynomial, exponential, etc.) with respect to certain elements such as the model size. One work in this direction is [8].

An additional problem is that we will need to create or, preferably, to count on big and distributed models to run the transformation. For that, we plan to make use of model repositories such as the case studies used in TTC (Transformation Tool Contest) [3]

Another important issue is to check the correctness of the parallel transformations, i.e., the transformations must be deterministic and the output models must be the same as in the sequential execution. For this, we plan to use the Tracts approach [11, 25] to test the implementation of the parallel execution engine by using a test set of models and model transformations.

Acknowledgments This work is funded by Research Project TIN2011-23795 and by the Universidad de Málaga (Campus de Excelencia Internacional Andalucía Tech). I would like to thank to my supervisors, Antonio Vallecillo and Manuel Wimmer, for guiding me throughout this work.

References

1. Amrani, M., Dingel, J., Lambers, L., Lúcio, L., Salay, R., Selim, G., Syriani, E., Wimmer, M.: Towards a model transformation intent catalog. In: Proc. of AMT'12, ACM (2012) 3–8
2. Burgueño, L., Troya, J., Wimmer, M., Vallecillo, A.: On the concurrent execution of model transformations with linda. In: Proc. of BigMDE'13, Budapest, Hungary, ACM (2013) 3:1–3:10
3. Cabot, J., Teniente, E.: A metric for measuring the complexity of ocl expressions. In: In Proc. of Model Size Metrics Workshop, co-located with MODELS'06. (2006)
4. Clasen, C., Didonet Del Fabro, M., Tisi, M.: Transforming Very Large Models in the Cloud: a Research Roadmap. In: Proc. of the 1st International Workshop on Model-Driven Engineering on and for the Cloud. (2012)
5. Cuadrado, J.S.: Towards a Family of Model Transformation Languages. In: Proc. of ICMT'12. Number 7307 in LNCS, Springer (2012) 176–191

[3] `http://planet-sl.org/ttc2013`

6. Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM Syst. J. **45**(3) (July 2006) 621–645
7. Espinazo-Pagán, J., Sánchez Cuadrado, J., Molina, J.G.: Morsa: A Scalable Approach for Persisting and Accessing Large Models. In: Proc. of MODELS'11. Number 6981 in LNCS, Wellington, New Zealand, Springer (2011) 77–92
8. Fernández-Ropero, M., Pérez-Castillo, R., Weber, B., Piattini, M.: Empirical assessment of business model transformations based on model simulation. In: Proc. of ICMT'12. Number 7307 in LNCS, Springer (2012) 137–151
9. Gelernter, D., Carriero, N.: Coordination languages and their significance. Communications of the ACM **35**(2) (1992) 96–107
10. GigaSpaces Technologies Ltd. GigaSpaces: (2013) `http://www.gigaspaces.com`.
11. Gogolla, M., Vallecillo, A.: *Tract*able model transformation testing. In: Proc. of ECFMA'11. Number 6698 in LNCS. Springer (2011) 221–235
12. Goldschmidt, T., Wachsmuth, G.: Refinement Transformation Support for QVT Relational Transformations. In: Proc. of MDSE'08. (2008)
13. Heckel, R., Küster, J.M., Taentzer, G.: Confluence of typed attributed graph transformation systems. In: Proc. of the First International Conference on Graph Transformation (ICGT). Number 2505 in LNCS, Springer (2002) 161–176
14. Jakumeit, E., Buchwald, S., Kroll, M.: GrGen.NET - The expressive, convenient and fast graph rewrite system. STTT **12**(3-4) (2010) 263–271
15. Jouault, F., Tisi, M.: Towards Incremental Execution of ATL Transformations. In: Proc. of ICMT'10. Number 6142 in LNCS, Springer (2010) 123–137
16. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: A model transformation tool. Science of Computer Programming **72**(1-2) (2008) 31–39
17. Kolovos, D.S., Rose, L.M., Matragkas, N., Paige, R.F., Guerra, E., Sánchez-Cuadrado, J., De Lara, J., Ráth, I., Varró, D., Tisi, M., Cabot, J.: A research roadmap towards achieving scalability in model driven engineering. In: Proc. of BigMDE'13, Budapest, Hungary, ACM (2013) 1–10
18. Lano, K., Kolahdouz-Rahimi, S.: The UML-RSDS manual (2012) `http://www.dcs.kcl.ac.uk/staff/kcl/uml2web`.
19. Mens, T., Gorp, P.V.: A Taxonomy of Model Transformation. Electr. Notes Theor. Comput. Sci. **152** (2006) 125–142
20. Muller, P., Fleurey, F., Jézéquel, J.: Weaving executability into object-oriented meta-languages. In: Proc. of MoDELS'05. Number 3713 in LNCS, Springer (2005) 264–278
21. Razavi, A., Kontogiannis, K.: Partial evaluation of model transformations. In: Proc. of the 34th International Conference on Software Engineering (ICSE), IEEE (2012) 562–572
22. Tisi, M., Jouault, F., Fraternali, P., Ceri, S., Bézivin, J.: On the use of higher-order model transformations. Number 5562 in LNCS, Springer (2009) 18–33
23. Tisi, M., Martinez Perez, S., Jouault, F., Cabot, J.: Lazy execution of model-to-model transformations. In: Proc. of MoDELS'11. Number 6981 in LNCS, Springer (2011) 32–46
24. van Amstel, M., Bosems, S., Kurtev, I., Ferreira Pires, L.: Performance in Model Transformations: Experiments with ATL and QVT. In: Proc. of ICMT'11. LNCS, Springer (2011) 198–212
25. Wimmer, M., Burgueño, L.: Testing M2T/T2M transformations. In: Proc. of MODELS'13. LNCS, Miami, FL, Springer (October 2013)
26. Wimmer, M., Martínez, S., Jouault, F., Cabot, J.: A Catalogue of Refactorings for Model-to-Model Transformations. Journal of Object Technology **11**(2) (2012) 1–40