Introducing Approximate Model Transformations

Javier Troya[1] and Antonio Vallecillo[2]

[1] Vienna University of Technology, Business Informatics Group, Austria
troya@big.tuwien.ac.at
[2] Universidad de Málaga, ETSI Informática, Spain
av@lcc.uma.es

Abstract. Model transformations dealing with very large models need to count on mechanisms and tools to be able to manage them. The usual approach to improve performance in these cases has focused on the use of concurrency and parallelization techniques, which aim at producing the correct output model(s). In this paper we present our initial approach to produce target models that are accurate enough to provide meaningful and useful results, in an efficient way, but without having to be fully correct. We introduce the concept of *Approximate Model Transformations*.

Keywords: Model Transformation, Approximation, Performance

1 Introduction

Model transformations are gaining acceptance as model-driven techniques are becoming commonplace. So far the community has mainly focused on the *correct* implementation of a model transformation, according to its specification [5,6,11], although there is an emergent need to consider other (non-functional) aspects such as performance, scalability, usability, maintainability and so forth. In particular, the study of the performance of model transformations is gaining interest as very large models living in the cloud need being transformed [1,2,9]. The usual approach to improve performance has focused on the use of concurrency and parallelization techniques.

In this paper we want to explore a different path. Our aim is to weaken the need to produce *correct* target models but *approximate* ones. Such approximate target models should be accurate enough to provide meaningful and useful results to users, but alleviate the need for the transformation to generate fully correct models—being able to produce such target models in much shorter time.

We call *Approximate Model Transformations* (AMTs) those model transformations that produce *approximate* target models. They are similar to the Approximation [10] or Randomized [7] Algorithms used in Computer Science and Operations Research.

This kind of model transformations are needed in various circumstances. The most obvious situation is when very large models have to be synthesized into much smaller models for decision making. We introduce a case study to present this problem. Let us consider a Wireless Sensor Network (WSN), where observation phenomena arrive and keep arriving as time moves forward. Consequently, the models grow with time (they are, in fact, *streaming* models [4]). Fig. 1 shows our metamodel for WSNs, developed with some ideas gathered from [8] and where four types of phenomena are

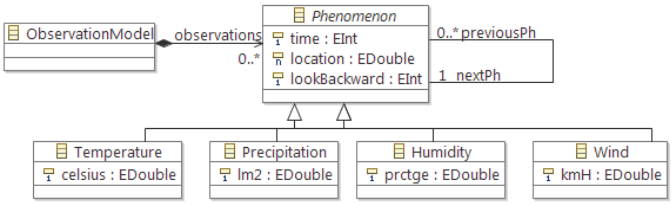

Fig. 1. Metamodel for observation phenomena in a WSN.

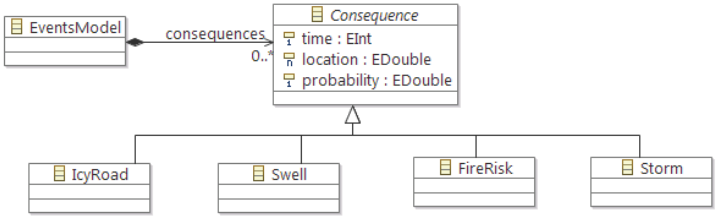

Fig. 2. Metamodel for defining consequences from observed phenomena.

identified. Each Phenomenon registers the time and location (in terms of the two terrestrial coordinates) where the observation takes place. Also, they keep record of the phenomena of the same type that was registered before and after the current one happened (previousPh and nextPh, respectively). As an inherent characteristic of streaming models, the latter relationship is not created when an object of type Phenomenon appears, but will be established later in time (when *future* phenomena appear). Reference previousPh points from a phenomenon to others phenomena of the same type that were obtained before, and only up to a certain point in time. Reference nextPh, in turn, points only to the next phenomena. Attribute lookBackward establishes the range of past observed phenomena – from the respective point in time of each phenomenon. Finally, each specialization of Phenomenon contains an attribute to keep the value obtained by the sensor node.

As explained in [8], to derive additional knowledge from semantically annotated sensor data, it is necessary to define and use a rule-based reasoning. In this way, when a group of sensor nodes provides information regarding for instance temperature and precipitation, then, using such rules, we can specify possible road conditions. One of these rules could be the following: if the temperature is less than 1 degree Celsius and it is raining, then the roads are potentially icy. To be able to express this kind of information in a model-based manner, we have created the metamodel shown in Fig. 2. It contains four different Consequences that are to be predicted according to the observed phenomena. Each consequence contains the time and location when and where it is predicted, as well as the probability that such consequence actually happens. Fig. 3 shows the preconditions in terms of observation phenomena, as well as the postconditions in terms of consequences, considered in our case study.

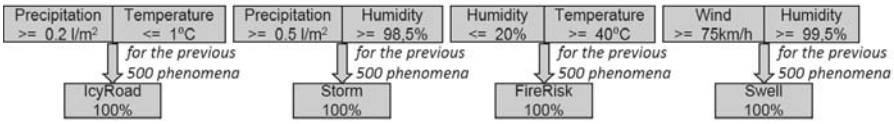

Fig. 3. Phenomena and Consequences.

In this paper, in the context of WSNs, we aim at reasoning over this kind of models not only considering the sliding window, i.e., the current information gathered from sensors that we have, but also taking into consideration that the information within such window can be potentially large. Thus, we focus on obtaining approximate predictions using AMTs by redefining some of the traditional operators used to traverse or query the models: allInstances(), select(), collect(), size(), etc. After this introduction, Sect. 2 presents our approach. Then, Sect. 3 describes an implementation and evaluation based on the case study and, finally, Sect. 4 draws some conclusions and future work.

2 Proposal

An AMT will likely not transform all the elements from the input model, for different reasons. For example, because they are not within the sliding window (e.g., they have already passed or have not arrived yet and there is no time to wait for them), or because the model is so large that we only work on a random sample of it [7]. This can result in some pending relationships in the output metamodel, what may make the output model not conform to its metamodel, and hence be *incorrect*. This kind of conformance is normally specified by means of constraints (also called contracts [3,5] in some contexts) that the model elements and their relations should respect.

In AMTs, it is also important the degree of *correctness* that the computed data have. In our case study, we consider that the probability of a consequence in a specific point in time is accurate according to certain rules as those shown in Fig. 3. Now, if the calculation of the probability is only based on a sample, and not on all the previous phenomena, the result may not be correct, just *accurate* enough for our purpose.

Approximate Operators. To elaborate our approach for defining AMTs, we apply a concept used in Randomized algorithms [7], namely *Random Sampling*. It is based on the idea that a small random sample from a population is representative of the population as a whole, and thus the properties of the sample can be used to determine some feature of the entire population.

Our proposal raises with the idea of redefining the common operators that current model transformation languages use to manage and operate with collections, such as allInstances(), collect(), select(), forAll(), etc. When transforming very large models, these operations become very expensive, performance-wise, because they have to traverse the whole model and deal with a large number of elements. If we reduce this number of elements, the transformation will execute faster. This is why we introduce a set of new collection operators, each one corresponding to an OCL collection operator.

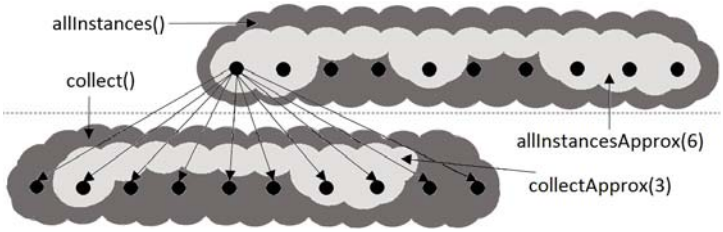

Fig. 4. Idea for Approximate Operators

The new ones end with "Approx" and incorporate an additional argument: an integer that represents the maximum number of instances the operator will process.

Fig. 4 shows this idea. In our case study, we may not need to obtain the predictions for all locations every time unit. Instead, we can randomly choose subsets of locations whose results can be extrapolated to surrounding areas, what corresponds to the upper part of the figure. Furthermore, since the creation of a consequence consists of the observation of a certain number of previous phenomena, we can also select a subset of them, what corresponds to the lower part of the figure. In this case, the values of the probabilities obtained in the consequences vary with respect to the originals, but they can be representative if they are close to the correct one.

3 Implementation and Evaluation

In our case study, the idea is to compute new consequences (conforming to the metamodel shown in Fig. 2) every time unit according to the phenomena observed by some sensor nodes (conforming to the metamodel shown in Fig. 1). This is potentially expensive, and the larger the number of sensor nodes deployed is, the longer it takes to process all data. We consider that the values of the measures gathered by two related sensor nodes (as shown in Fig. 3) has to be computed into a consequence if these nodes are not further than 10 away from each other, in Euclidean distance.

In the model we have considered, the slicing windows will contain 400 different locations, where the two locations which are collocated further away from each other have an Euclidean distance of 21.21. In fact, the sensor nodes in these locations have been randomly placed between coordinates $(0, 0)$ and $(15, 15)$. Sensor nodes of the four types in the metamodel (temperature, precipitation, humidity and wind) are distributed among these 400 locations, so that we have 100 points with each measure. Also, each Phenomenon keeps the history of the 500 previous phenomena of the same type. This means that attribute lookBackward has a value of 500 in our model.

We aim at obtaining the consequences for time 500. We have 200000 different phenomena in our input model, given how we have built it. Our transformation consists of one matched rule, four lazy rules and several helpers that perform the calculations. The lazy rules are called from the matched rule to build the EventsModel from an ObservationModel. Each lazy rule is used for obtaining consequences of the four different types, and each of them contains two inputs, as graphically depicted in Fig. 3. The way

Table 1. Accuracy for some consequences.

Location	Consequence	Prob. Original	Prob. AMT	Deviation
$(11, 10.2)$	Icy Road	78%	72.1%	5.9%
$(5, 10.8)$	Storm	47.2%	48%	0.8%
$(6.4, 3.2)$	Fire Risk	47.2%	49.4%	2.2%
$(8.1, 3.8)$	Swell	35%	28%	7%

we compute the consequences is merely a simplification of how it would be done in reality. Focusing for instance on icy roads, we say that the precipitation has to be equal or above $0.2l/m^2$ and the temperature equal or below 1 degrees Celsius, both during the previous 500 units of time, in order to have a 100% probability of icy roads. If not all the previous phenomena satisfy this condition, but some of them do, then we calculate the actual percentage by applying a simple cross-multiplication.

In the original transformation, a total of 28893 consequences are calculated in 111.2 seconds. An AMT has been constructed starting from the assumption that not all consequences have to be calculated every time unit, but calculating some of them is enough (upper part in Fig. 4). In the listing below we show how we make use of a selectApprox operation in order to take only 50 phenomena of each type (we show only the retrieval of Temperature phenomenon, it is the same for the others).

```
--Original Transformation
rule GenerateConsequences{
 from om : MM!ObservationModel
 using {
  ts : Sequence(MM!Temperature) = om.observations -> select (p |
           p.oclIsTypeOf(MM!Temperature) and p.time = 500);
(...)
--Approximate Model Transformation
(...)
  ts : Sequence(MM!Temperature) = om.observations -> selectApprox (50, p |
         p.oclIsTypeOf(MM!Temperature) and p.time = 500);
(...)
}
```

These operations are implemented now simply as helpers, although we plan to integrate them in the language as future work. Thus, since we choose a subset of the initial phenomena from which to obtain consequences, only 7362 are produced in the AMT, now in 22.1 seconds, what means a speedup of 5.03. We also realize an approximation according to the lower part of the figure. Now, instead of looking at the previous 500 phenomena of each type in order to calculate the consequence, we only consider half of the sample. As expected, the probability for the computed consequences is different than in the original transformation. Table 1 shows the values obtained in some locations in both approaches. As we can see, the deviation is not significant, so for this case study it is worth applying an AMT due to the gain in performance.

4 Conclusions and Future Work

This exploratory paper has presented a novel approach for the definition of Approximate Model Transformations (AMTs). We have explained its basic ideas and applied it in a case study. This paper presents just initial ideas that require deeper investigations at all levels. We were interested in exploring the possibility of defining AMTs, and our

initial results show that there are enough reasons to keep working on them. The work presented here does not pretend to be conclusive, or comprehensive, but to open the path for the modeling community to start working on it.

There are several open issues that we plan to address next. In the first place, we want to realize a comprehensive study on how accuracy can be measured depending on the particular scenarios, applications and model kinds. Second, we would like to provide formal and precise specifications for our approximate operators, and integrate them in the ATL language. Extending the approximate operators with a new argument that defines the maximum interval of time that we want the operator to execute is another line of work. Last, but not least, we need to study and develop methods for the appropriate design of AMTs. Although this will normally require a deep knowledge on the domain and the particular transformation scenarios, there is already a fair amount of work about the design of approximate and randomized algorithms that could be applicable in this context.

Acknowledgements. This work is partially funded by Research Project TIN2011-23795 and by the EC under ICT Policy Support Programme (grant no. 317859).

References

1. van Amstel, M., Bosems, S., Kurtev, I., Pires, L.F.: Performance in Model Transformations: Experiments with ATL and QVT. In: Proc. of International Conference on Model Transformations (ICMT). LNCS, vol. 6707, pp. 198–212. Springer (2011)
2. Burgueño, L., Troya, J., Wimmer, M., Vallecillo, A.: On the Concurrent Execution of Model Transformations with Linda. In: Proc. of the Workshop on Scalability in Model Driven Engineering (BigMDE 2013). ACM Digital Library (2013)
3. Cariou, E., Belloir, N., Barbier, F., Djemam, N.: OCL contracts for the verification of model transformations. Electronic Communications of the EASST 24 (2009)
4. Cuadrado, J.S., de Lara, J.: Streaming Model Transformations: Scenarios, Challenges and Initial Solutions. In: Proc. of International Conference on Model Transformation (ICMT 2013). LNCS, vol. 7909, pp. 1–16. Springer (2013)
5. Gogolla, M., Vallecillo, A.: *Tract*able Model Transformation Testing. In: Proceedings of the 7th European Conference on Modelling Foundations and Applications (ECMFA 2011). LNCS, vol. 6698, pp. 221–235. Springer (2011)
6. Guerra, E., de Lara, J., Wimmer, M., Kappel, G., Kusel, A., Retschitzegger, W., Schönböck, J., Schwinger, W.: Automated verification of model transformations based on visual contracts. Autom. Softw. Eng. 20(1), 5–46 (2013)
7. Motwani, R., Raghavan, P.: Randomized algorithms. ACM Comput. Surv. 28(1), 33–37 (1996)
8. Sheth, A., Henson, C., Sahoo, S.S.: Semantic Sensor Web. IEEE Internet Computing 12(4), 78–83 (2008)
9. Tisi, M., Perez, S.M., Choura, H.: Parallel Execution of ATL Transformation Rules. In: Proc. of MODELS 2013. LNCS, vol. 8107, pp. 656–672. Springer (2013)
10. Vazirani, V.V.: Approximation Algorithms. Springer (2003)
11. Wimmer, M., Burgueño, L.: Testing M2T/T2M Transformations. In: Proc. of MODELS 2013. LNCS, vol. 8107, pp. 203–219. Springer (2013)