Facilitating Reuse in Model-Based Development with Context-Dependent Model Element Recommendations

Lars Heinemann
Technische Universität München, Germany
heineman@in.tum.de

***Abstract*—Reuse recommendation systems suggest code entities useful for the task at hand within the IDE. Current approaches focus on code-based development. However, model-based development poses similar challenges to developers regarding the identification of useful elements in large and complex reusable modeling libraries. This paper proposes an approach for recommending library elements for domain specific languages. We instantiate the approach for Simulink models and evaluate it by recommending library blocks for a body of 165 Simulink files from a public repository. We compare two alternative variants for computing recommendations: association rules and collaborative filtering. Our results indicate that the collaborative filtering approach performs better and produces recommendations for Simulink models with satisfactory precision and recall.**

***Keywords*-model-based development; software reuse; recommendation system; data mining**

I. Introduction

Model-based development has gained increased attention due to its promise of improved productivity. Models become primary development artifacts and, in order to obtain a running system, the models are interpreted or executable code is generated from them. The main development and maintenance activities shift their focus from code to models. Instead of writing code, developers work with models represented in a domain specific language. The modeling languages for data flow and processing models, used, for instance, in the area of control systems, often have a notion of *reuse*. Model elements can be composed hierarchically and reuse occurs by using elements multiple times. Moreover, often the modeling language toolkit provides collections of reusable elements in form of libraries. Since the amount of reusable elements can grow large, just as with classical code-based development, it is a challenge to find a reusable element for a given task. For code based development, recommendation systems have been proposed to alleviate this. These systems suggest library elements, such as methods or classes, useful for the task at hand [1]–[3]. In this paper, we transfer this idea to the field of model-based development.

Mathworks Simulink[1] is a development environment for modeling and simulating dynamic systems which utilizes a block diagram notation [4]. Simulink has become a prevalent technology in the embedded systems domain, especially in the automotive field. TargetLink[2] is a C code generator for Simulink models that allows to automatically generate executable C code. Simulink models are constructed from *blocks*, the atomic functional units of the Simulink language. The Simulink development environment offers a large variety of predefined blocks, arranged in block libraries. To use these block libraries, the Simulink development tool suite provides a library browser that allows to navigate a hierarchical categorization of the blocks. The 16 standard Simulink libraries consist of 135 blocks. In addition to the provided libraries, Simulink can be extended with custom libraries. A number of additional libraries are provided by Mathworks as well as third-party vendors. These block libraries and the size of industrial Simulink models, which can grow as large as 20.000 model elements [5], motivate the adoption of recommendation systems for model-based development.

A. Problem

Existing work on recommendation systems focuses on code-based development. However, model-based development poses similar challenges to developers, regarding the identification of useful elements in large and complex modeling libraries. Even for experienced developers, finding the right element in large libraries for a given task can be challenging. Thus, new approaches are required to assist developers in effectively using modeling libraries and thereby facilitate reuse in model-based development.

B. Contribution

This paper introduces an approach for the context-dependent recommendation of elements from reusable modeling libraries during model based development. It uses data mining techniques to extract knowledge from existing models that is used to produce recommendations for unfinished models under development. We instantiate the approach for the Simulink modeling language and evaluate the quality of the recommendations with a case study using 165 model files from a public repository. We compare two variants for producing recommendations: association rules and collaborative filtering.

[1] http://www.mathworks.com/products/simulink/

[2] http://www.dspace.de/en/pub/home/products/sw/pcgs/targetli.cfm

II. Data Mining Background

Before we describe our approach, this section briefly introduces the data mining concepts this work is based on.

A. Association Rules

Association rules (ARs) are commonly employed for shopping basket analysis [6]. The goal is to identify regularities in transaction data which can be used, for instance, to better align the business to customer needs. A transaction (or shopping basket) is typically given as a set of purchased items where quantities are abstracted. An example is $\{nachos, dipping\ sauce, cola\}$. ARs are of the form $I \rightarrow j$, where I is a set of items and j is an item. An example is $\{nachos, dipping\ sauce\} \rightarrow cola$ denoting that customers who bought nachos and dipping sauce "typically" also bought cola. The Apriori algorithm [7] mines ARs from a set of shopping baskets. It computes frequent item sets in the shopping baskets that have a certain *support threshold*, *i. e.*, fraction of baskets in which the item set occurs. From the frequent item sets, association rules are built that have a desired *confidence threshold*. The confidence for an association rule $I \rightarrow j$ is given by the ratio between the number of baskets containing both I *and* j and the number of baskets that contain I. The ARs can then be used to recommend additional items for shopping baskets that they are *applicable* to. An AR is applicable to a given shopping basket if the left side of the rule is a subset of the shopping basket. The two threshold parameters of the algorithm influence the number and quality of the obtained ARs.

B. Collaborative Filtering

The basic idea of collaborative filtering (CF) is that users with similar preferences regarding items will rate other items similarly as well [8]. CF uses a database of users, items and a *like*-relation between users and items, to recommend items to users. Based on what a user already likes, similar users are searched and additional items that the similar users like are recommended. As an example, let us assume that we have data about which movies users like. In the data base, there exists a user u_1 who likes the movies {*The Time Machine, Star Wars, Moby Dick*}. For a user u_2 who likes the movies {*The Time Machine, Moby Dick, Back to the Future*}, we may recommend the movie *Star Wars* as u_2 showed similar interest for other movies.

A common approach for implementing a CF-based recommendation system is to encode the items which a user likes as a vector and define similarity among users as the similarity between the corresponding vectors in the vector space. A vector similarity measure that is often used is the *cosine similarity*, which corresponds to the cosine of the angle between the vectors. It is computed as follows:

$$cosine\_similarity(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| \times |\vec{b}|}$$

To obtain recommendations, the *k-nearest-neighbor* algorithm can be used. The k most similar users are determined according to the similarity measure. Their item preferences are aggregated and the set of recommended items is derived. The value of k influences the number and quality of the recommendations.

III. Approach

Our approach assumes that the modeling language has the concept of functional elements that can be used for composing models. We abstract a model as the set of distinct elements used by it. Hence, we do not consider in what quantities the elements are used or how they are interconnected. Our approach analyzes which distinct elements are used in the models of a training corpus in the *training phase*. In the *recommendation phase*, elements that are not used yet are recommended for unfinished models. We implemented two variants of the model recommendation approach: an AR-based and a CF-based variant.

A. AR-Based Recommendation System

Training Phase: We adopt ARs for model-based development by considering the set of distinct elements used in a model as a shopping basket and the individual elements as items. The mining process analyzes a corpus of training models by extracting the set of elements employed in the models. It uses the Apriori algorithm to mine association rules, which are stored in a file for later use in the recommendation phase.

Recommendation Phase: Using the mined association rules, the recommendation system suggests additional model elements for partial models. For this, the recommendation system is given the set of elements employed so far in an incomplete model. The recommendation system iterates over all association rules and recommends the associated element of each applicable rule if it is not yet employed in the model.

B. CF-Based Recommendation System

Training Phase: For the CF-based recommendation system, we consider the models as users and the model elements as items. Also here, we abstract how many times an element is used in a model. The training phase only consists of the extraction of the element usage for each model. We store a list of sets of model elements for later use in the recommendation phase.

Recommendation Phase: For a query, given as a set of model elements, the k most similar sets from the training phase are determined. All elements that the neighbors collectively use and that are not already used in the query model are returned as recommendations.

C. Instantiation for Simulink Models

Simulink models are a hierarchical composition of Simulink subsystems, whereby the atomic units are Simulink

Table I
NUMBER OF DISTINCT BLOCKS USED PER SUBSYSTEM

min	max	mean	p25	median	p75
2	16	4.72	3	4	6

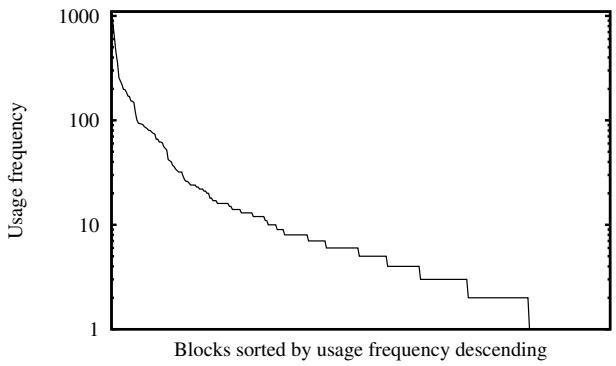

Figure 1. Log-scale diagram of block usage in study objects

blocks. For AR mining, each Simulink subsystem is considered as a shopping basket. Consequently, the ARs associate library blocks. As an example, the following AR means that models using the blocks $Gain$, $Integrator$ and $Constant$ typically also use the Sum block:

$$\{Gain, Integrator, Constant\} \rightarrow Sum$$

For CF, we consider Simulink subsystems as users and blocks as items. The block usage of a subsystem is encoded as an n-dimensional boolean vector, where each component refers to a specific block and thus n corresponds to the overall number of different blocks among all models. Within the vector, a component of "1" indicates that a particular block was employed (one or more times) in the model and a "0" means that it was not employed.

IV. CASE STUDY

A. Study Objects

We used as study objects 165 Simulink files downloaded from the Matlab Central File Exchange[3]. We excluded uses of the blocks $Inport$, $Outport$ and $SubSystem$ completely, since these elements do not provide functionality on their own and occur very often in the models. We also filtered models that used only one type of block, since no association rules can be mined from them. Thereby, we obtained 1103 subsystems which collectively used 335 distinct library blocks. Table I shows information on the number of distinct blocks used per subsystem.

To illustrate the recommendation problem, Figure 1 shows a log-scale diagram of the block usage frequency among the study objects. The distribution can be compared to the *long tail* effect in marketing. While there are few blocks that are used very frequently, there is a large amount of blocks that are used infrequently but in sum account for a large fraction of the usages. As an illustration, if we exclude the 20 most used blocks, still about 35% of all block usages employ none of the "popular" blocks. In other words, during modeling, in 35% of the cases, a modeler needs a "non-commodity" block and could thus benefit from a recommendation system.

[3]http://www.mathworks.com/matlabcentral/fileexchange/

B. Design and Procedure

We performed a 10-fold cross validation, *i. e.*, we conducted 10 evaluations where in each we used 90% of the 1103 subsystems as the training set and the remaining 10% as the test set. To evaluate the approach for a given subsystem in the test set, we created an artificial unfinished subsystem by randomly removing half of its blocks and queried the recommendation system with the remaining half. Depending on the recommendations made and the blocks actually employed (removed previously), we assessed the quality of the recommendations by measuring precision, recall and F-measure. Formally, these are computed as follows:

$$\text{precision} = \frac{correct\ recommendations}{total\ recommendations}$$

$$\text{recall} = \frac{correct\ recommendations}{actually\ employed\ blocks}$$

$$\text{F-measure} = \frac{2 \times precision \times recall}{precision + recall}$$

In addition to our approach, we evaluated a trivial baseline recommendation system, which always recommends a certain amount of the most frequently used blocks. For the AR-based variant, we chose, as a result of preliminary experiments, 1% as support threshold. We evaluated different values for the confidence threshold. For the CF-based variant, we evaluated different values for k.

C. Results

Baseline Recommendation System: Figure 2 shows the precision, recall and F-measure values for the baseline recommendation system. The results depend on the number of blocks recommended from the top used blocks. For a value of 3 recommended blocks, the recommendation system performs best and achieves an F-measure of 0.19. For this setting the precision is 0.17 and the recall is 0.21.

AR-Based Recommendation System: Figure 3 shows the results for the AR-based recommendation system for different values of the confidence threshold. As expected, the precision increases and the recall decreases with an increasing confidence threshold. For a confidence threshold of 0.4, the F-measure assumes the best value of 0.31. For this optimal setting, precision and recall are 0.32 and 0.30 respectively. Table II shows information on the number of recommendations returned per query for this setting.

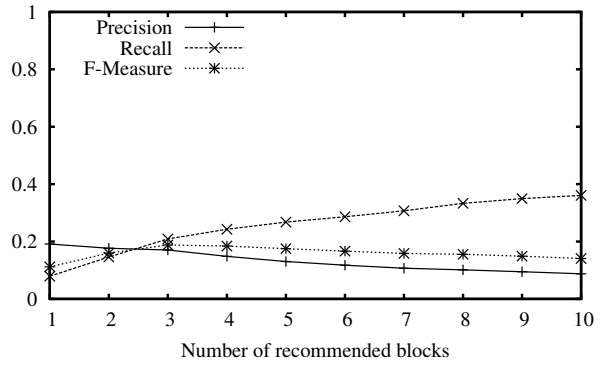

Figure 2. Baseline recommendation system

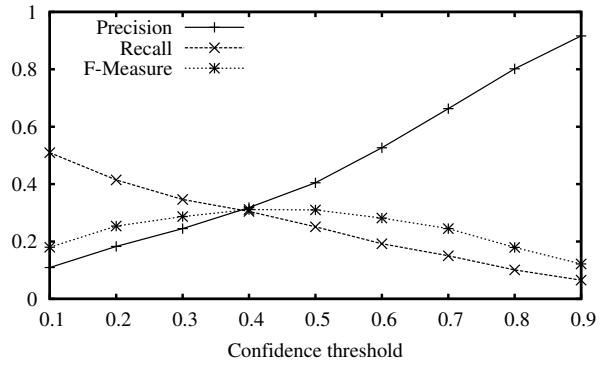

Figure 3. AR-based recommendation system

CF-Based Recommendation System: Figure 4 shows the results for the CF-based recommendation system for different values of k (number of neighbors considered). The diagram shows that for increasing values of k, the recall is increasing whereas the precision is decreasing. The F-measure has its optimum value of 0.56 for $k = 1$, where the precision is 0.66 and the recall is 0.49. Information on the number of recommendations returned per query for $k = 1$ is shown in Table II.

V. Discussion

The results show that both the AR-based and the CF-based variant of the recommendation system clearly outperform the baseline approach. The results also make the trade-off between precision and recall for both variants apparent, which depends on the chosen parameter values. While there is an optimum of the parameters with regards to F-measure, the variability allows to adjust the recommendations to a user's preference. For instance, an unexperienced user might want to rather obtain more recommendations while accepting to also receive potentially irrelevant recommendations.

Table II
Number of recommendations per query

	min	max	mean	p25	median	p75
AR (conf=0.4)	0	11	2.26	1	2	3
CF (k=1)	0	9	1.75	1	1	2

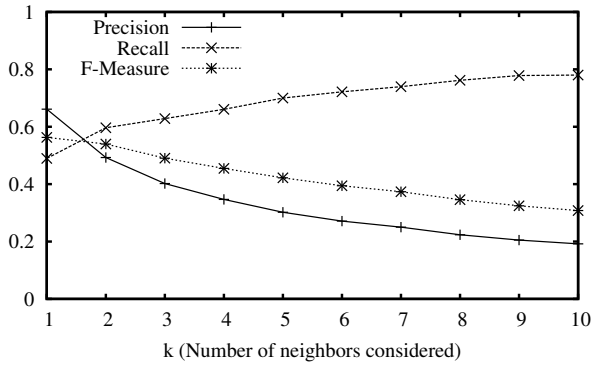

Figure 4. CF-based recommendation system

Moreover, it can be seen that the AR-based variant in general achieves better values for precision in comparison to the CF-based variant whereas for recall, the relation is the other way round. In terms of F-measure however, taking the trade-off between precision and recall into account, the CF-based variant outperforms the AR-based variant.

VI. Threats to Validity

A. Internal Validity

The *twinning problem* denotes duplicate or near duplicate values in the data set. If, during cross validation, one twin is in the training data set and the corresponding twin is in the test data set, better evaluation results are obtained. We mitigated this with a simple duplicate detection on the Simulink files. We excluded files whose content was identical to another file.

We performed an automated evaluation by predicting removed blocks from subsystems. We thus do not know how useful the recommendations would be perceived by a user. However, we assume that users may even consider blocks other than those eventually employed in a model as useful.

The evaluation assumed that half of the blocks of a subsystem were already employed to produce recommendations. However, a user would ideally like to get recommendations with less blocks already employed.

We determined the query set for the evaluation randomly. However, the resulting partial subsystem might not correspond to an intermediate state as it would occur during modeling. This can lead to an error in both directions, *i. e.*, better or worse recommendation quality during real use.

B. External Validity

It is unclear how the study objects are representative for all Simulink models. However, since the models of the case study collectively used a total of 335 distinct blocks, we assume a certain diversity among the study objects.

Since we restricted our evaluation to Simulink models, we do not know how the approach transfers to other model-based development approaches. We consider this an important direction for future work.

VII. Related Work

A. Recommendation Systems in Software Engineering

A general introduction to recommendation systems in software engineering is given by Robillard et al. in [9]. A number of approaches have been proposed for recommending API elements such as types and methods [1]–[3], [10], [11] or code examples [12], [13] based on the code being edited in the IDE. However, to the best of our knowledge, this is the first work to transfer the idea of recommendation systems to model-based development.

B. Content-based Model Search

In content-based model search, the task is to find models in a repository that are similar to a given query model (fragment). Therefore, as for our approach, a notion of similarity between (partial) models is required. Existing approaches [14], [15] use different graph-matching techniques to retrieve similar models. In contrast to these approaches, we use a very simple notion of similarity between models, abstracting completely from the number and interconnections of model elements, focussing only on the set of distinct model elements that are used.

VIII. Conclusion and Future Work

We presented an approach for recommending useful model elements during model-based development that assists developers in using large and complex modeling libraries. Our results show that the approach can produce recommendations for Simulink modeling with satisfactory precision and recall. We conclude that the transfer of recommendation system approaches to model-based development is a promising path to follow. As the work presented in this paper is still in early stages, a number of interesting open research questions remain.

A possible direction for future research is to experiment with different notions of similarity of (partial) models for collaborative filtering. One option would be to take into account other aspects of the model, such as the interconnections between the model elements.

Currently, the recommendations depend on *all* blocks employed in an unfinished subsystem. Thus, the recommendations are independent of where a new model element is to be inserted during editing. An alternative would be a more narrow context as given by a certain amount of predecessors of a given model element. It is an interesting open question, if this could lead to a higher precision.

We are also interested in applying our approach to other modeling languages as well as to evaluate our method for models from industrial systems.

Acknowledgement

The author is grateful to Benjamin Hummel and Andreas Vogelsang for inspiring discussions and helpful comments.

References

[1] M. Tsunoda, T. Kakimoto, N. Ohsugi, A. Monden, and K. Matsumõto, "Javawock: A Java Class Recommender System Based on Collaborative Filtering," in *SEKE05*, 2005.

[2] F. Mccarey, M. Cinnéide, and N. Kushmerick, "Rascal: A recommender agent for agile reuse," *Artificial Intelligence Review*, vol. 24, pp. 253–276, 2005.

[3] L. Heinemann, V. Bauer, M. Herrmannsdoerfer, and B. Hummel, "Identifier-Based Context-Dependent API Method Recommendation," in *CSMR'12*, 2012.

[4] J. Dabney and T. Harman, *Mastering Simulink*. Prentice Hall, 2004.

[5] F. Deissenboeck, B. Hummel, E. Jurgens, B. Schatz, S. Wagner, J. Girard, and S. Teuchert, "Clone detection in automotive model-based development," in *ICSE'08*, 2008.

[6] M. Bramer, *Principles of data mining*. Springer, 2007.

[7] R. Agrawal, R. Srikant *et al.*, "Fast algorithms for mining association rules," in *VLDB'94*, 1994.

[8] X. Su and T. Khoshgoftaar, "A survey of collaborative filtering techniques," *Advances in Artificial Intelligence*, vol. 2009, p. 4, 2009.

[9] M. Robillard, R. Walker, and T. Zimmermann, "Recommendation systems for software engineering," *IEEE Software*, vol. 27, no. 4, pp. 80–86, 2010.

[10] M. Bruch, M. Monperrus, and M. Mezini, "Learning from examples to improve code completion systems," in *ESEC-FSE'09*, 2009.

[11] S. Thummalapenta and T. Xie, "Parseweb: a programmer assistant for reusing open source code on the web," in *ASE'07*, 2007.

[12] R. Holmes and G. C. Murphy, "Using structural context to recommend source code examples," in *ICSE'05*, 2005.

[13] H. Zhong, T. Xie, L. Zhang, J. Pei, and H. Mei, "MAPO: Mining and recommending API usage patterns," in *ECOOP'09*, 2009.

[14] B. Bislimovska, A. Bozzon, M. Brambilla, and P. Fraternali, "Content-based search of model repositories with graph matching techniques," in *SUITE'11*, 2011.

[15] R. Dijkman, M. Dumas, and L. García-Bañuelos, "Graph matching algorithms for business process model similarity search," *Business Process Management*, pp. 48–63, 2009.