# Separating Local and Global Aspects of Runtime Model Reconfiguration

Frank Trollmann, Grzegorz Lehmann, Sahin Albayrak

DAI-Labor, TU Berlin
Faculty of Electrical Engineering
and Computer Science
Frank.Trollmann@dai-labor.de, Grzegorz.Lehmann@dai-labor.de,
Sahin.Albayrak@dai-labor.de

**Abstract.** There is a growing need for applications that are able to adapt themselves to the context of use. One promising approach for the adaptation of an application during its execution is the use of models at runtime. In this approach models of the application and its context of use are kept alive during the execution. The application can be adapted by reconfiguring the structure of these models.
Model Reconfiguration has local aspects as it handles the structure of a model and has to deal with its specific properties. It also possesses global aspects, as the joint reconfiguration of several models is required due to consistency considerations. This paper aims at solving possible conflicts between the global and the local aspects of Model Reconfiguration by introducing a distinction between two Levels of abstraction that enables the designer to separate and interrelate global and local aspects of Model Reconfiguration.

## 1 Introduction

There is a growing need for applications that are able to adapt themselves to the current context of use. Especially in dynamic and personalized areas like smart homes such adaptive applications are important as they can adjust to the user's specific needs as well as her environment. Adaptations trigger changes in the applications user interface or behavior. In [1] the need for adaptations as well as special challenges in this field of research is stressed.

Adaptability results in a growing complexity of software applications and their development. According to [2], one promising approach for dealing with this complexity is the models at runtime approach (also called models@run.time approach). This approach is similar to Model Driven Engineering (MDE) [3], where the development of a software application is accompanied by the creation and transformation of a set of models. The difference between these approaches is in their goal. MDE aims to generate the final application code from the intermediate models. The goal of the models@run.time approach is to keep the models alive at runtime. In this approach the running application results from an interpretation of these models.

In the models@run.time approach a running application consists of a set of models. These models represent different aspects of the application and its context of use. The applications user interface and behavior result from the interpretation of these models within a certain framework. The advantages of this approach for adaptations are twofold. First, the current state of the application and its context of use can easily be retrieved by querying the representing models. This is essential because the application is supposed to react to these states by adapting itself. Second, a change in the structure of one or more of these models automatically affects the execution of the application. Thus, the adaptation of an application can be achieved by changing the structure of its models. We call such a structural change a Model Reconfiguration.

Some adaptations of an application require the joint reconfiguration of several models. This can be necessary if an intended adaptation is within the scope of several models at once. Another reason for a joint reconfiguration of models is to maintain the consistency in models that partially overlap in the aspects they represent. According to this, Model Reconfiguration needs to have global aspects that enable the designer to express joint reconfigurations of several models.

A models@run.time framework may contain different models expressed in different modeling languages. Reconfiguration techniques are often specific to one modeling language. This allows them to optimally deal with the specific structural and behavioral properties of this modeling language. This means Model Reconfiguration also needs to have certain local aspects.

When implementing a framework for Model Reconfiguration, there can be a conflict of interest between these local and global aspects. In this publication we propose a distinction between a Reconfigurable Model and a Reconfiguration Model. This distinction serves to establish a separation of concern between the local and global aspects of Model Reconfiguration, which allows the designer to model and interrelate both.

The paper is structured as follows. First, in Section 2 the problems arising from the global and local aspects of Model Reconfiguration are subsumed. Reconfigurable Models are then discussed in Section 3. Afterwards, our description of the Reconfiguration Model is given in Section 4. In Section 5 existing approaches to Model Reconfiguration are introduced and related to the concepts introduced in this paper. Section 6 gives a conclusion and hints to future work.

## 2 Problem Statement

While authors are clear about the overall goals of Model Driven Engineering, the actual set of models required to build an application is far from fixed. One possible set of models is described in the CAMELEON Reference Framework [4]. Current approaches differ in the set of used models, as well as the modeling languages these models are described in.

The same goes for models@run.time approaches. Several possible runtime ensembles of models do exist and several modeling languages are used for describing

these models. Techniques for Model Reconfiguration are often local to a certain modeling language. This determines the local aspects of Model Reconfiguration.

These local aspects are important because Model Reconfiguration is tightly interwoven with the structure that is reconfigured. The advantage of a reconfiguration technique, specifically tailored to a modeling language, is that it can handle or preserve certain properties, specific to this modeling language. One example for such a specific approach is the Graph Transformation technique, analyzed in [5]. This technique can only be applied to P/T nets. Due to this exclusiveness it is able to preserve the firing behavior of these net.

It is also important to change the structure of several models at the same time. This enables the designer to treat the set of runtime models as a consistent whole. For example, the CAMELEON Reference Framework contains three different models for user interfaces: the Abstract, Concrete and Final User Interface Model. When these models are used at runtime they have to be kept consistent with each other. This can be best done by reconfiguring them jointly and thus ensuring consistency after each reconfiguration.

The set of runtime models within an application is not restricted to a fixed set of models. In addition, the models, used within one runtime ensemble are likely to be modeled in different modeling languages. This constitutes a potential conflict between global and local aspects of Model Reconfiguration. According to the local aspects it is possible that the designer chooses a different reconfiguration technique for each model in the runtime ensemble. These techniques are local to their modeling language and cannot be applied the other models. This is a problem for the global aspects of Model Reconfiguration which require a joint reconfiguration of the set of runtime models.

Our approach towards these potential problems is to separate the local and global aspects within two different components. The idea is to unite each model and its local reconfiguration as a so-called Reconfigurable Model. A component, called a Reconfiguration Model, steers the global reconfigurations. This model uses the Reconfigurable Models in order to accomplish this goal.

For this separation to work, the Reconfiguration Model should be able to abstract from the following properties of the Reconfigurable Models:

- *Model Kinds:* The Reconfiguration Model should be independent of the kinds of used models as well as their purpose in order to not restrict the designer to a fixed set of models.
- *Modeling Language:* The Reconfiguration Model should not be limited in the set of modeling languages it is able to reconfigure. This way the designer is free in her choice of modeling languages.
- *Reconfiguration Technique:* The Reconfiguration Model should be able to abstract from the used reconfiguration technique. This way the designer is free in her choice of reconfiguration technique.

In Sections 3 and 4 the concepts of Reconfigurable Model and Reconfiguration Model are introduced and discussed in detail.

Figure 1 shows an example for excerpts of a Task and User Interface model which are part of a runtime ensemble. The Task Model represents the applica-

tions structure of task and subtasks. The User Interface Model represents its user interface. These two models represent the user interface of a login window and its part of the task tree. The user is able to input username and password in parallel and then finish the task "Login" by clicking on the login button.

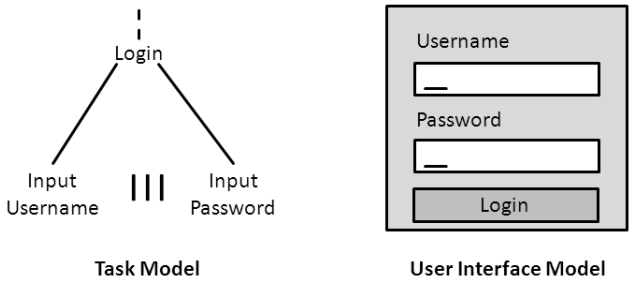


**Fig. 1.** Running Example: Two Models of a Login Mask

One possible adaptation of the user interface is to show the input of username and password in consecutive windows. A reason for this adaptation is the availability of different authentication protocols. In this case it is not certain that the user needs a password to log into the system. For example, he could also be identified by the MAC Address of his devices. In this case the input of a password becomes obsolete. Such an adaptation requires changes in the User Interface Model and the Task Model. The user interface has to be adapted to showing two windows. One to input the username and one to input the password. In addition the task model has to reflect the fact that "Input Username" and "Input Password" are now executed consecutive.

This example is oversimplified as the reconfiguration is really more complex than indicated here. The changes in the models are more complex due to the requirement to reflect more than one authentication method. In addition, other models have to be reconfigured to connect the new user interface and its execution logic. Nevertheless, this toy example already requires the joint reconfiguration of two models and will serve as a running example throughout the rest of the paper.

## 3 Reconfigurable Model

The notion of a Reconfigurable Model is introduced in order to encapsulate the local aspects of Model Reconfiguration. In each Reconfigurable Model the designer is able to concentrate on the structural changes of one model. In this scope she is able to choose a reconfiguration technique that suits her preferences. In this Section the notion of a Reconfigurable Model is defined and discussed.

A Reconfigurable Model is a model that can be reconfigured. In addition to the models structure it contains means for changing this structure. A scheme for a

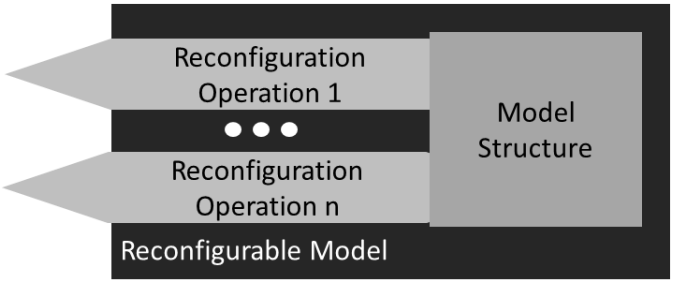


**Fig. 2.** Scheme of a Reconfigurable Model

Reconfigurable Model can be seen in Figure 2. A Reconfigurable Model consists of a Structure and a set of Reconfiguration Operations. The Reconfiguration Operations can be executed, resulting in a change of the structure. A more formal definition of a Reconfigurable Model can be seen in Definition 1.

**Definition 1 (Reconfigurable Model).** *A Reconfigurable Model r=(S,OPs) consists of a model S, determining the Structure of the Reconfigurable Model, and a set of Reconfiguration Operations OPs, that can be applied to change this Structure.*

The Structure $S$ of a Reconfigurable Model is not restricted. An arbitrary model conforming to an arbitrary meta model may constitute this structure. Reconfiguration Operations *OPs* represent ways to change $S$ in a way that it still conforms to its meta model. A more formal definition of a Reconfiguration Operation is given in Definition 2.

**Definition 2 (Reconfiguration Operation).** *A Reconfiguration Operation for a meta model MM is a function* $OP : M \rightarrow M$ *mapping one model, that conforms to MM to another one. M is the set of all models that conform to MM.*

A Reconfiguration Operation is a function that can be applied to the current Structure $S$ to generate a new Structure. This operation strictly acts within the set of models conforming to the meta model *MM* of *S*. Thus, the Reconfiguration Operations cannot violate the conformity to the meta model. A Reconfiguration Operation can be executed from outside of the Reconfigurable Model without any knowledge of *S*. For each Reconfiguration Operation *OP*, a Reconfiguration Endpoint *OP'* is available which automatically applies *OP* to *S*.

While modeling a Reconfigurable Model the designer has to provide the current Structure $S$ as well as the set of Reconfiguration Operations *OPs*. In principal, any model can be used for defining *S*, regardless of its modeling language. Based on this model and its modeling language, the designer then chooses the most suitable reconfiguration techniques to model *OPs*.

This process represents the standard case of producing a Reconfigurable Model. Other variations are also imaginable. For example the Reconfiguration

Operations could be generated automatically from another description of variability, like an enumeration of all possible structures.

At runtime, $S$ is used as the initial structure of the Reconfigurable Model. The Reconfiguration Operations can be executed in order to change this structure. These operations and their Reconfiguration Endpoints enable external models, like the Reconfiguration Model, to trigger changes in S.

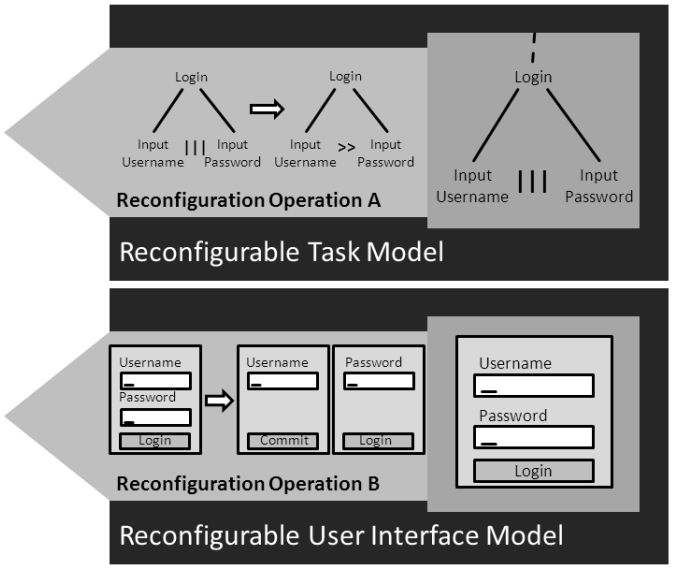


**Fig. 3.** Running Example: Reconfigurable User Interface and Task Model

In our Running example there are two models that need to be made reconfigurable. These are the Task and User Interface Model, introduced in Figure 1. In Figure 3 their reconfigurable versions are depicted. In both cases the structure $S$ consists of the models introduced in Figure 1. Each model contains one Reconfiguration Operation. In the Reconfigurable Task Model the two parallel tasks "Input Username" and "Input Password" can be made consecutive. The Reconfiguration Operation of the Reconfigurable User Interface Model splits the login window and distributes the input elements for username and password.

In the Figure it is not mentioned how these Reconfigurations are implemented. The techniques, used in both models are independent from each other and can be chosen by the designer. Due to the graphical representation of a user interface she might decide to use a form of Graph Transformation in the Reconfigurable User Interface Model. For changing one temporal operator in the Reconfigurable Task Model she might choose an action as provided by the KerMeta environment. These two Reconfiguration Operations describe the changes, required for our Running Example. However, on the level of Reconfigurable Mod-

els it is not possible to interrelate these two model changes. This is the purpose of the Reconfiguration Model, introduced in the next Section.

## 4 Reconfiguration Model

In the previous section Reconfigurable Models as an encapsulation of the local aspects of Model Reconfiguration, are described. This section introduces the notion of a Reconfiguration Model. This model builds upon the definition of Reconfigurable Models and reflects the global aspects of Model Reconfiguration.

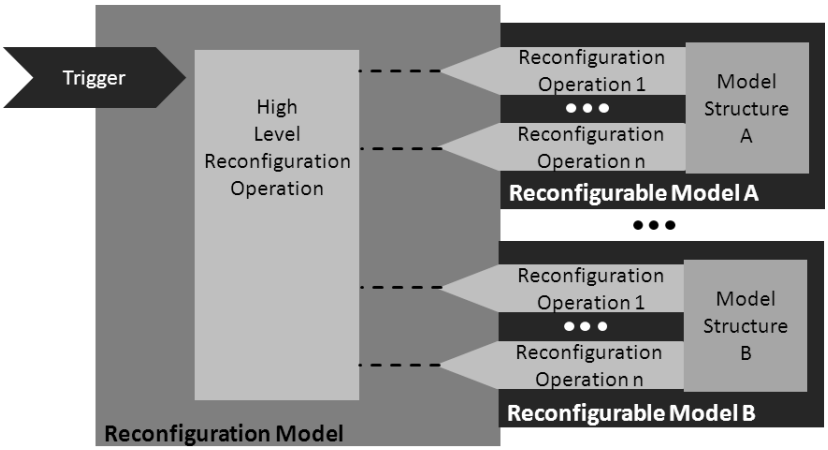


**Fig. 4.** Scheme of a Reconfiguration Model

A scheme for a Reconfiguration Model can be seen in Figure 4. A Reconfiguration Model contains High Level Reconfiguration Operations. Each of them is connected to a trigger, which is responsible for determining when to execute this Operation. The Reconfiguration Model can adapt a set of Reconfigurable Models. This is done by executing the Reconfiguration Endpoints of their Reconfiguration Operations.

The triggers serve as Guards for executing the High Level Reconfiguration Operations. Whenever a trigger fires the High Level Reconfiguration Operation is executed and calls the Reconfiguration Operations it requires.

The designer models the High Level Reconfiguration Operations to describe complex joint reconfigurations of several models. The structural changes in each model are accomplished by calling the Reconfiguration Operations, provided by their Reconfigurable Model. This represents the global aspects of Model Reconfiguration. Inside this global description of reconfiguration logics she is able to abstract from the properties mentioned in Section 2:

– *Model Kinds:* The Reconfiguration Model can work with any Reconfigurable Model regardless of its inner implementation or purpose. Therefore, it is independent of the actual set of models used at runtime.
– *Modeling Language:* The structure that is reconfigured is hidden within the Structure $S$ of a Reconfigurable Model. The Reconfiguration Model only triggers the Reconfiguration Operations for changing this structure and does not touch the structure directly. Thus, the Reconfiguration Model can work independent of the Modeling Language.
– *Reconfiguration Technique:* The Reconfiguration Model only needs a reference to the Reconfiguration Operations in order to execute them. It does not have to know how they are implemented. Thus, it can work independent of the reconfiguration technique used to describe the Reconfiguration Operations.

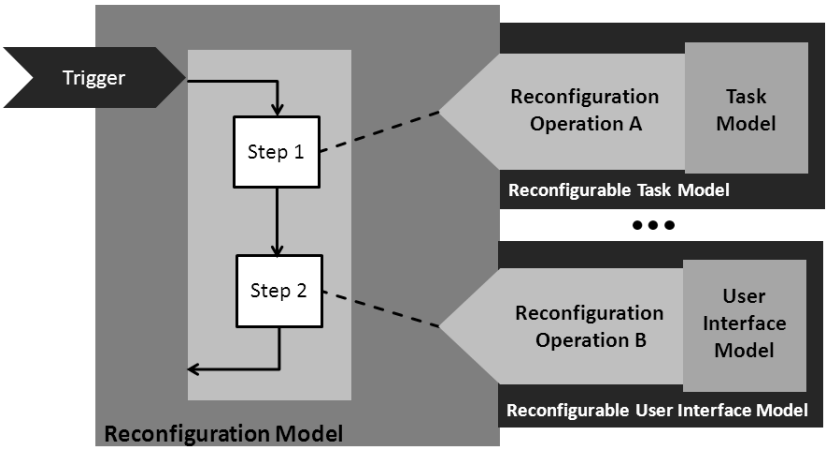


**Fig. 5.** Running Example: The Reconfiguration Model

A Reconfiguration Model for our running example is depicted in Figure 5. This Reconfiguration Model uses the Reconfigurable Task Model and Reconfigurable User Interface Model depicted in Figure 3. It contains one High Level Reconfiguration Operation, which consists of two steps. In the first step the Task Model is reconfigured, using Reconfiguration Operation A. In the second step Reconfiguration Operation B of the Reconfigurable User Interface Model is triggered. This High Level Reconfiguration Operation executes all changes discussed in our running example.

The purpose of this publication is to propose the differentiation between local and global aspects of Model Reconfiguration and their separate handling by introducing two levels of abstraction. The complete definition of both components is still work in progress. Although a High Level Reconfiguration Operation is depicted as a series of consecutive steps in the running example, we do not believe

this to be the final or best solution. Some hints on possible implementations are given in the remainder of this section.

There are several possible ways to implement a High Level Reconfiguration Operation. For example, a visual language could be used for describing the workflow of application of the Reconfiguration Operations. Languages like UML - Statecharts, Flowcharts or Petri Nets could be utilized for this task. Several lessons can also be learned from the field of Graph Transformation, where several means for high level transformation logic are proposed. Imperative programming languages, like Java or C++, could also be utilized.

Before deciding on one of these alternatives a detailed analysis of the properties and control structures, required for a comfortable modeling of high level reconfigurations has to be carried out. In the next section, related work in the field of Model Reconfiguration is discussed. This work is then related to the local and global aspects of model reconfiguration and the concepts introduced in this paper.

## 5 Related Work

Several approaches towards Model Reconfiguration have been implemented. This section serves to introduce some of these approaches and interrelate them to the local and global aspects identified in this paper and our notion of Reconfigurable Model and Reconfiguration Model.

One of the most recognized techniques for reconfiguring models is Graph Transformation. A Graph Transformation rule searches and substitutes one occurrence of a pattern within a graph with another one. The concrete syntax of several modeling languages can be described as a graph. For this reason a variation of a Graph Transformation technique is an obvious choice for structural adaptations in these languages. A variety of applications for Graph Transformations to dynamic systems can be found in [6]. Although neither of these applications is specially applied to the reconfiguration of models at runtime, they all contain an initial structure that is reconfigured using graph transformation rules. This is very similar to our notion of a Reconfigurable Model.

Several specific Graph Transformation languages do exist. These languages are dedicated to a certain modeling language and can only be applied to models within this language. For example, in [5] a Graph Transformation approach for rewriting P/T nets is introduced. This technique preserves the firing behavior of P/T nets. This shows the capability of specific Graph Transformation languages to preserve properties of the transformed models and their structure. A specific Graph Transformation Language can be a good choice of a reconfiguration technique to describe Reconfiguration Operations.

In [7] this specific Graph Transformation language for P/T nets is applied to model a flexible emergency scenario. In this publication, the initial scenario is modeled using a P/T net and the possible changes to this scenario are modeled as a set of Graph Transformation rules, specific for P/T nets. This setup is very similar to our notion of a Reconfigurable Model. The P/T net can serve as the

current Structure $S$ and the set of Graph Transformation rules are similar to our Reconfiguration Operations $OPs$.

In [8], Schürr studies approaches towards building programmed graph replacement systems from Graph Transformation rules. He also proposes his own approach towards unifying these approaches. The purpose of programmed graph replacement systems is to provide means for defining complex schemes of reconfiguration out of Graph Transformation rules and thus enable the designer to take a global view on Graph Transformation. However, an abstraction from the concrete reconfiguration technique or modeling language was out of scope for Schürr. For this reason programmed graph replacement systems do not make these abstractions. Nevertheless, we consider this publication to be a valuable source of inspiration for the design and implementation of High Level Reconfiguration Operations.

USIXML [9] uses Graph Transformation techniques in a more general scope. In this framework all models are described in XML. This format is used as the basis for Graph Transformation. This enables a transformation between different models, used for backward and forward engineering. The approach towards Graph Transformation taken in USIXML is also an interesting one for Model Reconfiguration as it enables the designer to describe several models as one joint XML file and then reconfigure them jointly. This approach captures certain global aspects. However, it can only be applied to models that are described within an XML structure. Thus, it is not general enough to satisfy our requirements from Section 2.

Graph Transformation is not the only concept that has been tested within the scope of Model Reconfiguration. In [10] Morin et Al. describe their approach towards modeling adaptive systems using models and aspects. The system is specified as a set of aspect models. They are weaved into one runtime model, which represents the whole running application. The system is adapted by reconfiguring the aspect models and weaving a new runtime model. This newly woven runtime model is then compared to the old one and a script for transforming the old into the new one is generated. In this publication model reconfiguration also clearly has local and global aspects. Reconfigurations are specified for each aspect model but are then woven into one runtime model. Global consistency can be checked by specifying a set of consistency constraints. However, this can only serve to check consistency after the reconfiguration. In our opinion a way for specifying how two aspects are reconfigured jointly in order to preserve their consistency is still required.

The meta modeling language KerMeta [11] can also proof as a useful tool for Model Reconfiguration. The purpose of this language is to provide a language that is able to model the structure and behavior of a modeling language. The behavior is modeled by an action language. This way the designer of a modeling language can model the structure and behavior of this language jointly. This approach can also proof interesting for model reconfiguration as structural changes of such models can also be described by this action language. The KerMeta

action language can be used as a technique for implementing Reconfiguration Operations.

In this section we introduced a selection of approaches towards runtime reconfiguration of models. None of these approaches were explicitly able to model all local and global aspects of Model Reconfiguration. However, several similarities between these approaches and the components and separation introduced in this paper have been found. This leads us to expect that the separation and components we introduced, even given their current level of abstraction, capture many of the aspects, also addressed by these publications and are a good starting point for further research towards a universal Reconfiguration Model.

## 6 Conclusion and Future Work

This paper proposes a separation between a Reconfigurable Model, which is a model that offers certain Reconfiguration Operations that can be executed at runtime, and a Reconfiguration Model, which is responsible for triggering and steering the reconfigurations in all models used at runtime. The aim of this separation is to provide an approach to Model Reconfiguration that captures local and global aspects. Local aspects are strongly interwoven with the used models and modeling languages. Global aspects concern the interrelation of several models and their joint reconfigurations.

The Reconfigurable Model reflects local aspects of Model Reconfigurations and enables the designer to model Reconfiguration Operations that are close to the used modeling languages. In the Reconfiguration Model the designer can take a global view on Model Reconfiguration and interrelate the reconfigurations of different models.

Several existing approaches have been analyzed regarding their capability to capture the global and local aspects of Modeling Reconfiguration. Although none of the analyzed approaches had the flexibility to deal with all our requirements, they had several similarities to our approach.

In the near future the notions of Reconfigurable Model and Reconfiguration Model have to be further detailed. For example, the current definition provides no means for expressing additional application conditions for Reconfiguration Operations. For this step several sources of inspiration have been identified within the related work.

In Future Work we also plan to define a specific language for describing Reconfiguration Models. In Section 4 some ideas on how the components within this Reconfiguration Model can be implemented are given. These sources of inspiration have to be analyzed for their actual usability before a decision towards the final implementation can be made. In addition to a language for expressing such high level reconfiguration operations, a set of control structures, like conditional or repeated application of rules has to be defined and formalized.

Additionally, we plan on defining and executing a case study with the defined reconfiguration language as a proof of concept.

## References


1. Coutaz, J.: User interface plasticity: Model driven engineering to the limit! In: ACM, Engineering Interactive Computing Systems (EICS 2010) International Conference. Keynote paper., ACM publ. (2010) 1–8 Keynote paper.
2. Blair, G., Bencomo, N., France, R.B.: Models@ run.time. Computer **42**(10) (2009) 22–27
3. Schmidt, D.C.: Model-driven engineering. IEEE Computer **39**(2) (2006)
4. Calvary, G., Coutaz, J., Thevenin, D.: A unifying reference framework for the development of plastic user interfaces, Springer-Verlag (2001) 173–192
5. Ehrig, H., Habel, A., Kreowski, H.J., Parisi-Presicce, F.: Parallelism and concurrency in high-level replacement systems. Math. Struct. in Comp. Science **1** (1991) 361–404
6. Blostein, D., Schürr, A.: Computing with graphs and graph rewriting. Technical report, FACHGRUPPE INFORMATIK, RWTH (1997)
7. Hoffmann, K., Ehrig, H., Padberg, J.: Flexible modeling of emergency scenarios using reconfigurable systems. In: Proc. of the 10th World Conference on Integrated Design & Process Technology. (2007) 15 CDROM.
8. lim: Programmed graph replacement systems. In: In Rozenberg, G. (Ed.), Handbook on Graph Grammars: Foundations, World Scientific (1997) 479–546
9. Limbourg, Q.: Multi-Path Development of User Interfaces. PhD thesis, Université Catholique de Louvain, Institut d'Administration et de Gestion (IAG), Louvain-la-Neuve, Belgium (2004)
10. Morin, B., Barais, O., Nain, G., Jezequel, J.M.: Taming dynamically adaptive systems using models and aspects. In: ICSE '09: Proceedings of the 31st International Conference on Software Engineering, Washington, DC, USA, IEEE Computer Society (2009) 122–132
11. alain Muller, P., Fleurey, F., marc Jézéquel, J.: Weaving executability into object-oriented meta-languages. In: in: International Conference on Model Driven Engineering Languages and Systems (MoDELS), LNCS 3713 (2005, Springer (2005) 264–278