



System Modelling and its Decomposition into Local Views by Using Graph Transformation

N. H. ARIJO⁺⁺, Z. U. A. KHUHRO*, K.T. PATHAN, N. CHANNA**, S. A. MEMON

Institute of Information and Communication Technology, University of Sindh, Jamshoro.

Abstract: Modelling a system as a global system (whole world system) has usually scalability problems, so we are modelling a system through graph transformation and decomposing it into local views. These views can be integrated into global system in any tool supporting modular integration to avoid scalability problems. Which mainly occur while analysing a system through state space exploration.

Keywords: Modularity, Decomposition, Synchronization, Graph Transformation, System Modelling

1.

INTRODUCTION

In this paper, we will discuss our methodology for system modelling and decomposition into local views. We propose three approaches, monolithic, top-down and bottom-up. Modelling a system, as a single graph transformation system is the monolithic approach; decomposing a system into views and then synchronizing them is the top-down approach and composing a system from its subsystems is the bottom-up approach.

Graph Transformation

Graphs are an intuitive and very powerful formalism for modelling system architecture and its behaviour showing the conceptual structure and the functionality of a system. Type graphs depict the architecture of a system and graph transformation rules model the behaviour. The concept of graph Transformation was introduced in the early 1970s as a generalization of term rewriting to graphs and Chomsky's (string) grammars (Ehrig, 2006).

Type and Instance Graphs

Similar to class diagrams in object-oriented models or entity relationship diagrams in data models, we use type graphs to restrict the set of admissible graphs (Ehrig, 2004). One common means to restrict the shape of an object is to prescribe a type for the object.

A type graph (TG) is a graph whose nodes represent node types and whose edges represent edge types. A graph that is typed over TG also called instance graph over TG, is a graph G equipped with a graph morphism type $G : G \rightarrow TG$ that assigns a type to every node and edge in G . A type graph morphism is a morphism that preserves the typing of graphs. An edge type in TG represents a structural relationship among

nodes of TG-typed graphs. This is because, due to the typing morphism, edges of an edge may only connect nodes of the node types that are incident to the edge type in TG. A node type can be compared to a class and an edge type can be compared to an association in a UML (Unified Modelling Language) class diagram (Noraida, 2007). Type graphs can be represented by UML class diagrams whereas instance graphs can be represented by corresponding UML object diagrams.

Attributed Type Graphs

Attributed graphs (Lara, 2007) are used for modelling data-intensive systems such as object-oriented systems. Attributes of a graph behave similarly as attributes of an object oriented system. Attributed graphs are those graphs in which nodes and edges can be assigned attributes of some data types. Node attributes can be used to store additional information in a node. As known from object-oriented languages, an attribute consists of a name and a data value. In the context of typed graphs, we have to declare the attributes belonging to a certain node type by their name and data type in the type graph. In a corresponding instance graph, every instance of the node type can carry its own values for these attributes.

In connection with graph transformations, attributes can be used as special labels that identify certain nodes along a sequence of transformation steps. They are also useful to restrict the applicability of transformation rules to situations in which involved nodes have specific attribute values. Attributed graphs can occur at both levels, as type graphs and as instance graphs. In type graphs, the attributes are in fact attribute declarations, and their values are in fact sort names that determine possible values to be assigned to the attributes in an instance graph. Similarly to the

⁺⁺Corresponding author N. H. Arijo email: niaz_arijo@hotmail.com Cell.No.+923360234323

*Institute of Mathematics and Computer Science, University of Sindh, Jamshoro.

**Institute of Business Administration, University of Sindh, Jamshoro

terminology for database systems, we call the set of all possible values that can be assigned to attributes a domain. The values belonging to the domain are partitioned into disjoint sorts representing different basic data types. A subset of the value domain consists of variable names, which can be assigned to attributes instead of concrete values. Since we want to declare node attributes in a type graph by using sort names as attribute values, type graphs are attributed over the set of sort names.

2. METHODS AND MATERIALS

2.1. Monolithic Approach

In the monolithic approach (**Fig. 1**), we model a system as a single graph transformation system. We first define the architecture of a system by means of a type graph. Then we define a start graph and a set of rules. The monolithic approach is demonstrated by our case study (Niaz, 2011), which is modelled in GROOVE (Rensink, 2004) and translated into PEPA (Tribastone, 2009) for analysis.

There are four steps in the monolithic approach.

- (1) System Modeling as Graph Transformation
- (2) Generation of the Labelled Transition System (LTS)
- (3) Transforming the LTS into a CTMC (Continuous Time Markov Chain)
- (4) Performance Analysis

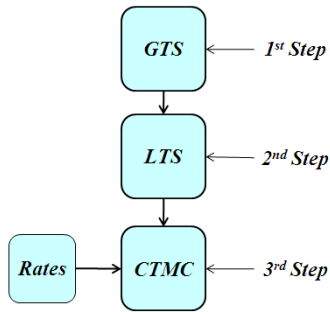


Fig. 1: Monolithic approach

2.2. Top-down Approach

Many system verification approaches based on state space exploration face scalability problems. We mitigate these problems by separating the model into different views. Decomposing a system into views, we then synchronize them over shared labels in the top-down approach (**Fig. 2**). It is composed of the following six steps.

- (1) System Modelling as Graph Transformation
- (2) Generation of Local Views
- (3) Generation of Labelled Transition Systems

- (4) Transforming the LTS into a CTMC
- (5) Synchronization of Local Views
- (6) Performance Analysis

The model is decomposed into different views; and these views' LTSs are later synchronized in either PEPA or PRISM over shared labels.

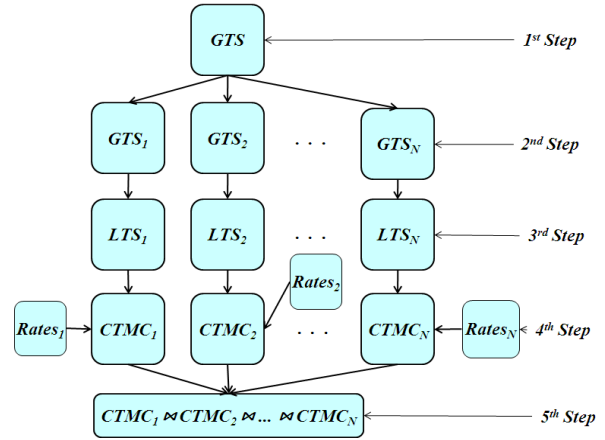


Fig. 2: Top-down modularity approach

Bottom-up Approach

Composing a system from its subsystems is the bottom-up approach (**Fig. 3**). In this approach, we have to identify subsystems or modules, which are perspectives or parts of a system and their common interface over which subsystems cooperate. The bottom-up approach requires a composition of individual subsystems, which share a common interface. It consists of the following five steps. Most of them are analogous to the monolithic and top-down approaches.

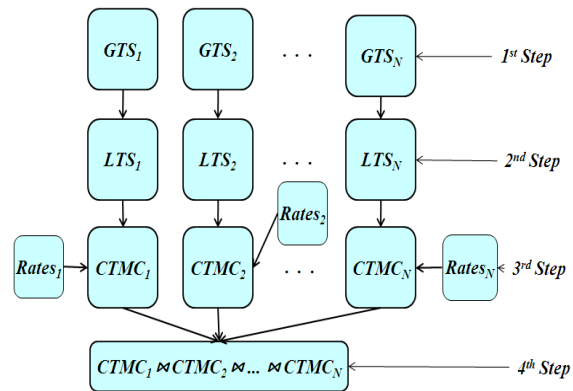


Fig. 3: Bottom-up modularity approach

- (1) Modelling of Views
- (2) Generation of the Labelled Transition System
- (3) Transforming the LTS into a CTMC
- (4) Synchronization of Local Views
- (5) Performance Analysis

3. **RESULTS AND DISCUSSIONS** **System Modelling as Graph Transformation**

In typed graph transformation systems, we define the structural model of a system as a type graph, which gives the architecture and imposes constraints on instance graphs. We then define a start graph (an instance graph) and a set of rules, which are applied to the start graph non-deterministically to produce the state space of the system.

Generation of the labelled transition system (LTS)

A labelled transition system (LTS) (Bert, 2000; Brinksma, 2001) gives us all the possible evolutions of a system throughout its life cycle. It is generated when all applicable rules are applied on a start graph, and all subsequent graphs, at all possible matches. In a labelled transition system, states represent isomorphism classes of graphs and transitions represent rule applications. Instantiated rule signatures serve as labels.

Transforming the LTS into a CTMC

We translate the LTS into CTMC (Baier, 2000; Brinksma, 2001), as GROOVE (Rensink, 2004) does not support stochastic analysis of a system such as steady-state analysis, passage time analysis etc. At the same time, we get another advantage from CTMC (Baier, 2000) for synchronizing individual views (subsystems) to compose a global model. We can integrate individual CTMCs and synchronize them over shared labels (rule names) to compose resulting CTMC, which is bisimilar (Aceto, 2011) to the CTMC of the global model. In this way, we get the same behaviour of the global model and avoid scalability problems, which arise in the verification of large systems. PEPA provides various performance analysis techniques; but it lacks support for large files; where as PRISM (Kwiatkowska, 2005; Kwiatkowska, 2009) has support for large files.

Modelling of Views

Graph transformation does not allow modelling a system as composition of subsystems. Systems are modelled as whole-world/global models. By defining a common interface among subsystems, we achieve system's modularity.

Generation of Local Views

In GROOVE (Graphs for Object-Oriented Verification) (Rensink, 2004) a system is modelled as a single graph transformation system. Therefore, in system analysis there are usually scalability problems. We suggest and demonstrate to model a system as composition of views (subsystems). We have developed a tool that generates local views from a global system based on the views' type graphs. For a decomposition of a system, we identify subsystems (views) on the base of the core functionality they are providing and defining

projections of the subsystems' type graphs out of the global system's type graph.

Types of nodes and edges that are required by the functionality of the view are retained and all other node types and edge types that are not used by the view are removed from the view's type graph. Often these subsystems are not disjoint, but share a common interface over which they co-operate. Rules and instances (start graphs) are then reduced according to the type graphs of local views by our View Generator tool that generates views on the basis of their local type graphs. Projected rules that are reduced to identities in one view and completely remain in the other view are removed from a view, while rules that partially remain are kept in the given views.

Synchronization of Local Views

In PEPA views are represented as individual sequential components. These sequential components are represented by initial states. These sequential components are merged together and are co-ordinated by the PEPA cooperation operator, e.g., $P1 \bowtie Q1$, while in PRISM views are represented as individual modules and PRISM synchronizes them over shared labels implicitly. PRISM also uses a CSP-based operator for synchronization, i.e., $M1 || M2$ (Dong, 2006).

Performance Analysis

Once we have transformed the GROOVE LTS into a PEPA or PRISM model, we can get stochastic analysis results supported by PEPA or PRISM. In PEPA we can analyse non-functional properties of mobile applications such as steady-state probabilities and throughput. Steady-state analysis determines the long-term probability for the system to be in a certain state and throughput analysis gives the long-term frequency of its actions (rule labels).

In PRISM we can analyse the steady state of a system. As PRISM supports continuous stochastic logic, where we can define properties and verify them, for example, the probability of arriving at the destination within a certain period of time, and the time within which an incident has occurred.

4. **CONCLUSION**

When we analyse a system through state space exploration there is usually a state space explosion problem. Therefore, we are modelling a system as composition of subsystems (modules) in GROOVE, and generate their LTSs, which are transformed by our created tool into CTMCs. These CTMCs could be synchronized in any tool supporting modular integrity just as PEPA and PRISM. In addition, we can also perform stochastic analysis of systems in these tools.

REFERENCES:

- Aceto L., A. Ingolfssdottir, J. Srba (2011) The Algorithmics of Bisimilarity, Advanced Topics in Bisimulation and Coinduction, Cambridge University Press, 2011
- Baier, C., B. H. Haverkort, J. P. Katoen, (2000) Model checking continuous-time markov chains by transient analysis, *Computer Aided Verification, Springer Verlag, Berlin*, **1855**, 358-372.
- Bert, D., C. F. Cave, (2000) Construction of finite labelled transition systems from B abstract systems, *In Integrated Formal Methods, Springer Berlin Heidelberg*, 235-254.
- Brinksma, E., H. Hermanns, (2001) Process algebra and markov chains, *Lectures on Formal Methods and Performance Analysis, Springer, Heidelberg, Berlin*, **2090**, 183-231.
- Dong, J. S. H. Ping, S. Jun, Z. Xian, (2006) A reasoning method for timed CSP based on constraint solving, *Formal Methods and Software Engineering, Springer-Verlag Berlin Heidelberg*, **4260**, 342-359.
- Ehrig, H., P. Ulrike, T. Gabriele, (2004) Fundamental theory for typed attributed graph transformation, *Lecture Notes in Computer Science, Springer Berlin Heidelberg*, **3256**, 161-177.
- Ehrig, H., K. Ehrig, U. Prange, and G. Taentzer, (2006) Fundamentals of Algebraic Graph Transformation, *Springer, Heidelberg*, 373.
- Kwiatkowska, M., G. Norman, and D. Parker, (2005) Probabilistic model checking in practice: Case studies with PRISM, *ACM SIGMETRICS Performance Evaluation Review, ACM*, **32**(4), 16-21.
- Kwiatkowska, M., G. Norman, and D. Parker, (2009), PRISM: Probabilistic model checking for performance and reliability analysis, *ACM SIGMETRICS Performance Evaluation Review, ACM*, **36**(4), 40-45.
- Lara, J. D, R. Bardohl, H. Ehrig, K. Ehrig, U. Prange, T. Gabriele, (2007) Attributed graph transformation with node type inheritance, *Theoretical Computer Science, Elsevier*, **376**(3), 139-163.
- Niaz A., R. Heckel, M. Tribastone, and S. Gilmore (2011) Modular performance modelling for mobile applications. In *Proceedings of the 2nd ACM/SPEC International Conference on Performance engineering (ICPE '11)*. ACM, New York, NY, USA.
- Noraida, H. A., S. Zarina, I. Sufian, (2007) A design of an Assessment System for UML class diagram, *International conference on computational science and its Applications, (ICCSA 2007), IEEE*, 539-546.
- Rensink, A., (2004) The GROOVE simulator: A tool for state space generation, *Applications of Graph Transformations with Industrial Relevance (AGTIVE), Lecture notes in Computer Science, Springer Verlag, Berlin*, **3062**, 479-485.
- Tribastone, M., A. Duguid, S. Gilmore, (2009) The PEPA Eclipse Plug-in, *ACM SIGMETRICS Performance Evaluation Review, ACM*, **36**(4), 28-33.