Advanced School on Visual Modelling Techniques

Participant Talks Abstracts

Modelling Tools:

Claudia Ermel: Tiger

Harmen Kastenberg: GROOVE

Christian Köhler: EMF Model Transformation

Ákos Horváth: The VIATRA2 Model Transformation Framework

Joel Greenyer: Reconciling TGGs with QVT

Felix Klar: Stratification meets Triple Graph Grammar

Antti Kervinen: How to Write Test Models for Symbian Devices

Scientific Session:

Raimundas Matulevicius: Comparison and Integration of Goal Modelling Languages

Benjamin Braatz: Adhesive HLR Transformation Systems

Marion Murzek: Structural Patterns for the Transformation of Business Process Models

Mikkel Bundgaard: Typed Polyadic Pi-calculus in Bigraphs

Troels C. Damgaard: Bigraphs and bigraphical reactive systems

Object Oriented and Rule-based Design of Visual Languages using Tiger

Tool Demonstration given by Claudia Ermel*
*Institut für Softwaretechnik und Theoretische Informatik
Technische Universität Berlin, Germany

Abstract

In this presentation we present the state-of-the-art of the TIGER environment for the generation of visual editor plug-ins in ECLIPSE, with the focus on its *Designer* component, a visual environment for object oriented and rule-based design of visual languages.

Domain specific modeling languages are of growing importance for software and system development. Meta tools are needed to support the rapid development of domain-specific tool environments. The basic component of such environments is a domain-specific visual editor. A visual language (VL) definition based on a meta model in combination with syntax rules defining syntax-directed editor commands is used in TIGER *(Transformation-based Generation of Environments)* to generate a corresponding visual editor. On the one hand, a visual language definition captures the visual symbols, links and relations of the domain specific modeling language (the alphabet); on the other hand, a syntax graph grammar defines precisely which editor operations are allowed and restrict the visual sentences of the VL to correct diagrams.

TIGER combines the advantages of precise VL specification techniques using graph transformation concepts with sophisticated graphical editor development features offered by the Eclipse Graphical Editing Framework (GEF). Using graph transformation at the abstract syntax level, an editor command is modeled in a rule-based way by just specifying the pre- and post-conditions of each command. The application of such syntax rules to the underlying syntax graph of a diagram is performed by the graph transformation engine AGG. TIGER extends AGG by a concrete visual syntax definition for flexible means for visual model representation. From the definition of the VL, the TIGER *Generator* generates Java source code. The generated Java code implements an Eclipse visual editor plug-in based on GEF which makes use of a variety of GEF's predefined editor functionalities. Thus, graphical layout constraints are defined and solved with efficient Java methods.

Based on an alphabet of finite automata we show how a visual language can be designed by defining the concrete syntax of the visual language and graph transformation rules for syntax directed editing of automata in the generated editor plug-in.

The TIGER environment may be downloaded at http://tfs.cs.tu-berlin.de/tigerprj.

GROOVE: Model Checking Graph Production Systems

Harmen Kastenberg

August 29, 2006

Combining the theory of both graph transformations and model checking opens new opportunities for specifying and verifying complex software or hardware systems in a natural and intuitive way. By modelling the system's states as graphs and its dynamic behaviour as graph transformation rules, we can generate a graph transition system containing all possible configurations of the system explicitly while keeping track of how each configuration can be reached.

The GROOVE Tool consists of two GUI components, namely the Editor and the Simulator. The former can be used to specify graphs and graph transformation rules; the latter is able to generate graph transition systems from specified graph production systems. The Simulator provides different strategies of generating the graph transition systems, ranging from a user-controlled step-by-step strategy to fully automatic exploration strategies generating the entire state space or only specific parts of it (e.g. linear or barbed exploration). GROOVE also offers a command-line fashion of the Simulator, the so-called Generator. The Generator is useful in cases when you are only interested in the final states of the graph production system, e.g. when performing model transformations, since no time is spend on graph-rendering.

The ultimate goal in the GROOVE project is to apply the well-founded theory of graph transformations for the verification of object-oriented systems. Many explicit state model checking approaches use bit vectors to represent the system's states. Unfortunately, that kind of representation does not extend smoothly to systems in which the states contain values from a domain other than primitive types, such as reference values commonly used in object-oriented systems.

In GROOVE, we currently apply a CTL model checking algorithm on state spaces generated using graph transformations. Because of the internal graph structure of states it is possible to handle the dynamic character of OO-systems more naturally as when using bit vectors. Furthermore, state space reduction techniques like symmetry reduction are included in the graph formalism in the notion of isomorphic graphs. We plan to investigate to what degree confluence properties of graph transformation systems as well as quantification of graph transformation rules can help reducing the size of state spaces.

Visual Model Transformation for EMF

Christian Köhler[1], Günter Kuhns[1], Enrico Biermann[1], Gabriele Taentzer[1], Karsten Ehrig[2], Eduard Weiss[1]

[1] Department of Computer Science, Technical University of Berlin, Germany,

{jaspo,bunjip,enrico,gabi,eduardw}@cs.tu-berlin.de

[2] Department of Computer Science, University of Leicester, UK,

karsten@mcs.le.ac.uk

Although there already exist proposals for EMF model transformations, a graph-based approach, where model transformations can be defined in a visual, rule-based manner has not been considered yet. We provide[1] a model transformation framework for EMF that consists of a graphical editor, an interpreter and a compiler that generates Java code. In-place model transformations for EMF are defined through transformation rules. These rules consist of a left-hand side (LHS), a right-hand side (RHS), possible negative application conditions (NACs) and mappings between these so called *object structures*. An object structure is a set of possibly linked and/or attributed objects, which are typed by certain EMF models. The left-hand side of a rule stands for the structural preconditions that must be fulfilled to apply the rule. Accordingly a right-hand side describes the result (or postconditions) of a rule. Negative application conditions are defined in the same way and describe structural conditions that must not be fulfilled to apply the rule. Objects

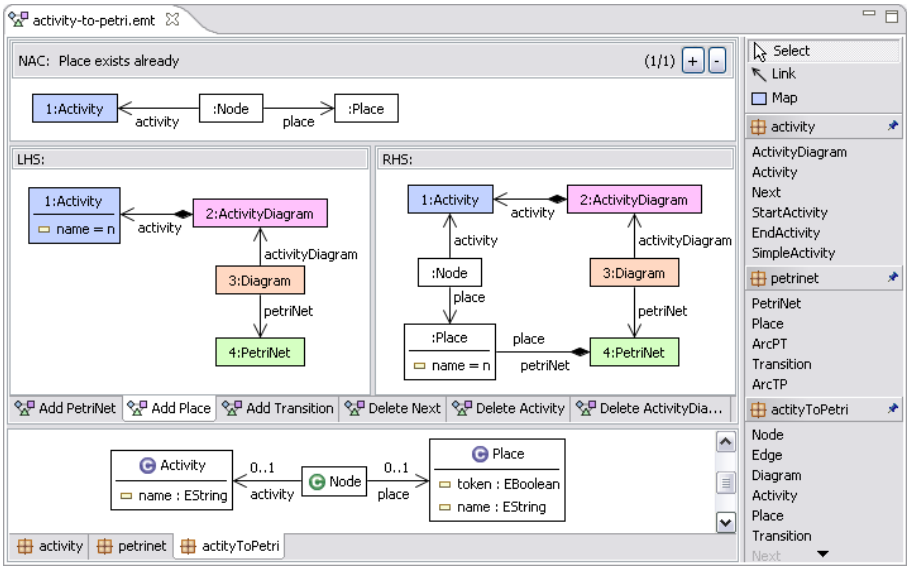

Figure 1: Transforming Activity diagrams to Petri nets.

in the LHS of a rule can be mapped to objects in the RHS and also to objects in the NACs. These mappings define which objects should be created, modified or deleted during rule application. The editor visualizes mappings by coloring the mapped objects in the same way. Further, attributes of objects in a rule can be modified by assigning Java expressions to them, which are evaluated during rule application.

[1] http://tfs.cs.tu-berlin.de/emftrans

The VIATRA2 Model Transformation Framework*

Ákos Horváth Dániel Varró

Dept. of Measurement and Inf. Systems
Budapest Univ. of Technology and Economics
Magyar Tudósok krt. 2, Budapest, Hungary, H-1117
{ahorvath,varro}@mit.bme.hu

Abstract

The VIATRA2 model transformation framework (available as an Eclipse plugin) primarily aims at designing model transformations, to support the precise model-based system development with the help of invisible formal methods. Formal methods are hidden by automated model transformations projecting system models into various mathematical domains (and preferably, vice versa).

VIATRA2 has chosen to integrate two popular, intuitive, yet mathematically precise rule-based specification formalisms namely: (i) graph transformation (GT) and (ii) abstract state machines (ASM) to manipulate graph based models. VIATRA2 also facilitates the separation of transformation design (and validation) and execution time. During design time, the VIATRA2 interpreter takes such MT descriptions and executes them on selected models as experimentation. Then final model transformation rules can be compiled into efficient, platform-specific transformer plugins for optimal execution.

*This work was partially supported by the SENSORIA European project (IST-3-016004)

Reconciling TGGs with QVT

Joel Greenyer

Software Engineering Group, Department of Computer Science,

University of Paderborn

jgreen@mail.uni-paderborn.de

Triple Graph Grammars (TGGs) are a convenient formalism to declaratively specify model transformations. The graph based nature and visual notation of TGGs are an intuitive way to specify relations between models. To investigate further potentially useful extensions or improvements to TGGs, this diploma thesis [Gre06] compared TGGs with QVT. QVT (Query/Views/Transformations) is the upcoming standard for model transformations in the context of Model Driven Architecture (MDA) which is issued by the Object Management Group (OMG). The standard is the result of selecting and merging different model transformation technologies, aiming at the practicability and the reuse of other related OMG standards. Because this comparison revealed many similarities between TGGs and QVT, a consequent question was to which extent QVT could be implemented by TGGs.

QVT is based on the Meta Object Facility (MOF) standard, which specifies the structure of models and metamodels as the foundation of MDA. A wide spread implementation of this specification is the Eclipse Modeling Framework (EMF) and so, for the purpose of this thesis, a flexible and extendable TGG interpreter for the transformation of EMF models was implemented as an Eclipse plug-in. Now, a transformation specified in QVT can be performed by the TGG interpreter by firstly translating the QVT rules into corresponding TGG rules which are then applied on the models.

The graph grammar based nature of TGGs allows to apply many verification techniques from the field of graph grammar theory. So, it is for example possible to prove the confluence of a TGG rule set or the correctness of the transformation result. Such verification techniques will become increasingly important in model driven software development. Furthermore, with the mapping from QVT to TGGs, these techniques would become accessible to QVT as well.

The current implementation of the TGG interpreter allows to translate not all, but the most important QVT language constructs into TGGs. The TGG interpreter design, however, prepares for extensions that would make this mapping yet more complete. If needed, further application specific operations may be plugged into the TGG interpreter. Specifically, an approach is presented to integrate the Object Constraint Language (OCL) into TGGs. This would then provide TGGs with an expressive power equal to QVT, which is extensively relying on OCL to describe model patterns. However, there is a tradeoff when extending TGGs by arbitrary expressions or operations, because it is rather the simplicity which is valuable for formal verification techniques.

Comparing TGGs with QVT revealed useful concepts that are now partly adopted by TGGs. But, this comparison also showed that TGGs yet have general advantages over QVT. TGGs provide, for example, more flexibility to express relations between model patterns. In many cases, this allows a more efficient transformation rule design. Now, there are many interesting subjects for future research. An alignment of TGGs with further QVT principles could yet improve the efficient use of TGGs in practice. For this purpose, concrete transformation and tool integration scenarios should be inspected. Furthermore, it could be investigated which verification techniques could aid most effectively during the design and processing of TGG transformations.

References

[Gre06] J. Greenyer: *A Study of Model Transformation Technologies: Reconciling TGGs with QVT*. University of Paderborn, July 2006. Master/Diploma thesis supervised by E. Kindler and R. Wagner

Stratification meets TGG

Felix Klar
Real-Time Systems Lab
Institute of Computer Engineering
Darmstadt University of Technology, Germany
Felix.Klar@es.tu-darmstadt.de

Today's large software systems have reached such a level of complexity that a single architectural view is not sufficient any more to appropriately capture their high-level architecture, detailed design, and low-level realization. Architecture stratification is an approach that connects multiple views on a single system with refinement translations, so that each view completely describes the whole system on a particular level of abstraction [KGK06].

Basic support for architecture stratification is realized by the tool SPin which is a plugin for the CASE-Tool Fujaba. SPin provides an annotation metamodel, which allows to enrich abstract syntax graphs (e.g. UML models) with abstract information and a transformation engine, which is able to transform (enriched) graphs by applying transformation rules to them. SPin only comes with a small set of predefined rules, but also provides the ability to define custom rules. They can be specified using Story Driven Modeling (SDM), which is a part of Fujaba. Transformation rules typically consist of a pattern matching and a transformation part.

To be able to support both forward and backward navigation between different levels of abstraction, two rules have to be specified—one for concretion (refinement rule) and one for abstraction (abstraction rule). In general abstraction rules are more difficult to implement than refinement rules, because more complex patterns have to be matched in a lower abstraction level. Backward navigation would be a lot easier, if additional information would be available, that define which elements were involved in creating the more concrete level. Those information can be established, e.g. by creating links, which connect elements from the more abstract with elements from the more concrete level, while applying a refinement rule.

For this purpose we are investigating the usage of the "Triple Graph Grammar" graph rewriting approach (TGG). TGGs are intended to support the specification of interdependencies between graph-like data structures on a very high level and they allow us to record additional information about the transformation process [Sch94], e.g. to establish correspondence links between left-hand side and right-hand side, in our case: abstract and concrete level.

References

[KGK06] Kühne, T.; Girschick, M. & Klar, F.: "**Tool Support for Architecture Stratification**", in: Mayr, H.C. & Breu, R. (eds.): GI (pub.), Proceedings of the Modellierung 2006, Vol. 82, LNI pp. 213-222, Innsbruck, Tirol, Austria, 22.-24. March 2006

[Sch94] Schürr, A.: "**Specification of Graph Translators with Triple Graph Grammars**", in: Tinhofer, G. (eds.): Springer-Verlag (pub.), Proceedings of the 20th International Workshop on Graph-Theoretic Concepts in Computer Science, Vol. 903, Lecture Notes in Computer Science pp. 151-163, Heidelberg, June 1994

How to Write Test Models for Symbian Devices

Antti Kervinen
Tampere Univ. of Technology, Institute of Software Systems
P.O.Box 553, FI-33101 Tampere, FINLAND. (ask@cs.tut.fi)

One of the biggest problems in taking model-based testing into practice seems to be the difficulty of creating a test model, that is, an executable formal description of the behaviour of the system under test. However, we have noticed that model-based testing would have many benefits compared to current test practices in our domain, that is, testing Symbian S60 smart phones through the user interface (UI). We try to overcome the difficulty of modelling in our domain by designing a domain-specific visual modelling language.

Our domain has many features which heavily affect the modelling. To mention some:

- There can be several applications running in the background, but only one of them controls the UI.
- The execution of an application can be interrupted by user actions, alarms, incoming calls and messages, for instance.
- Some applications interact directly, some indirectly through shared resources and data. Thus applications should be tested also simultaneously, not only one after another.
- Different phones contain different sets of applications.
- The UIs of applications change more often than their high-level functionality.

The listed features raise the following questions. How to model different interruptions and interactions of applications installed in the devices? On the other hand, how to isolate the volatile parts of the behaviour (changes in the UI and in the combinations of installed applications) to ease the maintenance of the test ware.

Our domain-specific modelling language is based on process algebra on LTSs. The basic idea is to write, or more precisely to draw, one high-level LTS for every application. Each of these LTSs defines the behaviour of an application without UI details. For example, an high-level LTS could specify that after starting Camera application one can take a photo, change camera settings, activate Gallery application or quit.

For every high-level LTS there are one or more low-level LTSs. They convert high-level actions to sequences of executable events in the UI. For example, a low-level LTS could specify that in a certain phone one can take a photo by either pushing a button or selecting "Capture" in a menu. Another low-level LTS could define how the photo can be taken in another phone. By separating high-level actions from low-level UI events we try to reuse the same high-level LTSs in a range of devices where the applications have the same functionality but may have different UIs.

The communication between high-level LTSs is possible with two sets of actions. The first set is used for requesting and giving permissions, usually for using shared resources or data. The second set contains actions for agreeing on which LTS is active in the sense that the corresponding application controls the UI. High-level LTSs communicate also with their low-level LTSs so that the high-level actions become refined to UI events. This communication is limited to actions "start/end high-level action X".

To generate a test model for a device the tester needs to select the high-level LTSs which are drawn for the applications installed in the device (or which of the applications he wants to test). Then the tester chooses the corresponding low-level LTSs. All the following steps are automatic. First, the model-based test tool runs a trivial (transition-splitting) transformation for certain transitions in the high-level LTSs. Then the tool reads the actions of all LTSs and generates rules for parallel composition, that is, decides which actions should be executed synchronously and which without synchronisation. Finally, the LTSs are composed in parallel on-the-fly during the test run.

More details in
A. Kervinen, M. Maunumaa, T. Pääkkönen, and M. Katara: "Model-based testing through a GUI". FATES 2005, July 2005, LNCS 3997, Springer-Verlag.

Comparison and Integration of Goal Modelling Languages

RAIMUNDAS MATULEVIČIUS AND PATRICK HEYMANS

Computer Science Department, University of Namur, Belgium
{rma, phe}@info.fundp.ac.be

Information systems (IS) are cornerstones of most human activities. Requirements define what an IS should do and the circumstances under which it is required to operate. *Goal-modelling languages* (GMLs) used during requirements engineering (RE), are the most prominent modelling approach, helping to deal with the complexity of the IS development. GML abstraction mechanism is based on the concepts of *goal*, goal *decomposition*, and goal *assignment* to *agent*. Focus on stakeholder's goals elicits not only of *what* the new system should do, but also *why* (i.e. rationale) the system is built; it provides a more stable scope for the developed IS. Literature defines a host of GMLs (such as KAOS, i*, GRL, Tropos, GBRAM, NFR and Lightswitch) and their variants, that have proved extremely valuable tools in a great number of situations. However, due to fragmentation of research [1], we have not yet observed a widespread adoption of GMLs by practitioners. This is regrettable since RE is where GMLs are expected to have the highest payoff. The more people work with one particular language, the more their thinking is influenced by this language, and their awareness of those aspects of the world that do not fit in, may consequently be diminished thus resulting in incomplete specification of the problem. For the types of problems that fit well with one modelling language, neglecting features that are not covered may even have a positive effect, because it becomes easier to concentrate on the relevant issues. However, it is hard to know what issues are relevant. Different issues within a problem situation may be relevant for different people at the same time, however not supported by the same GML. The survey [1] stresses the importance of more integration efforts to obtain a stronger GML that takes advantage of the many streams of goal-oriented research.

In this work we propose to yield a comparison and integration of GMLs. The languages are suggested to be analysed at the coarse- and fine-grained levels. The overall objective is to compare and integrate GMLs, and result with the *integrated GML*. The research includes five major activities: 1) *Language comparison* has the purpose to evaluate the GMLs on the coarse-grained level. Our approach includes the application of the semiotic quality framework [2] that might be strengthened with other quality standards suggesting the means to achieve the quality goals. 2) *Language constructs evaluation* defines the best evaluated GMLs at the fine-grained level using the UEML approach [3]. 3) *Language integration* has the purpose to investigate the integration possibilities of the GMLs evaluated at the second activity. 4) *Evaluation of tools* is expected to result with the tool assessment that would highlight the requirements for the prototype tool supporting the integrated GML. 5) Development and validation of the *prototype tool for integrated GML* is the fifth activity resulting with the prototype tool that would facilitate the application of the integrated GML.

We believe that such an integrated GML and the corresponding prototype tool would aim to accelerate the adoption of GMLs by practitioners. The expected results of this investigation would contribute with (1) a thorough systematic scientific investigation, comparison and integration of GMLs; and (2) an integrated and tool-supported GML for RE. The expected long-term benefits of goal-oriented language analysis are improvement of the quality and cost of the RE process and hence of IS. We also hope to drive the research community towards a more rigorous way to define and extend (goal) modelling languages.

References

[1] Kavakli E., Loucopoulos P. (2005). Goal Modeling in Requirements Engineering: Analysis and Critique of Current Methods. Krogstie J. *et. al.* (eds), Information Modeling Methods and Methodologies, Idea Group Publishing, 2005, pp. 102–124.

[2] Krogstie J. (2001). Using a Semiotic Framework to Evaluate UML for the Development for Models of High Quality. Siau K., & Halpin, T. (eds.). *Unified Modelling Language: System Analysis, Design and Development Issues*, IDEA Group Publishing, 89-106.

[3] Opdahl, A. L. (2006). The UEML Approach to Modelling Construct Definition. *Accepted at the Interoperability for Enterprise Software and Applications Conference* (*I-ESA'06*), Bordeaux, France.

Adhesive HLR Transformation Systems

Benjamin Braatz

SeGraVis-School, 10 September 2006

In this talk a new framework for the definition of semantic domains, which is a combination of transformation systems and adhesive high-level replacement (HLR) categories, is proposed. This framework is aimed at providing semantics for complex modelling techniques, where structural and behavioural aspects are modelled in an integrated way and, hence, traditional semantic domains such as algebras or labelled transition systems do not provide sufficient expressiveness.

A transformation system (see [GR04]), which is a concept for the integration of heterogeneous software specifications, consists of a data space with data states and data transformations and a control graph with control states and control transitions. Both, the data space and the control graph, are objects in the category of transition graphs, such that the category of all transformation systems over a given data space is a slice category as shown in Fig. 1.

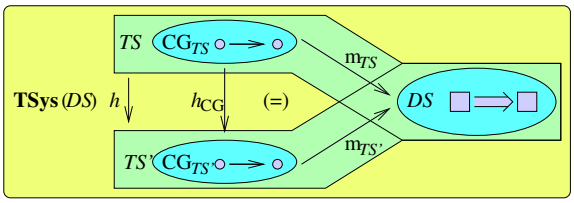

Figure 1: Category of transformation systems

In our approach, the data space as well as the control graph are defined by adhesive HLR systems (see [EEPT06]), which are a generalisation of graph transformations to objects in an arbitrary category. Hence, the instantiation of our framework is done by providing an adhesive HLR system for the data space, where the objects are the data states and the rules specify the data transformations, and a second adhesive HLR system, where the objects are control states and the rules describe the control transitions.

One notable difference to other approaches in this direction is that the rules of these adhesive HLR systems are fixed for the whole modelling technique. They are designed to interpret or execute given models and thereby provide a formal semantics for them.

The framework shall be instantiated for object-oriented systems to allow the definition of a formal semantics for UML and similar techniques. A first draft of this instantiation can be found in [BK06].

References

[BK06] Benjamin Braatz and Andreas Rayo Kniep. Integration of object-oriented modelling techniques, 2006. Draft version available from `http://tfs.cs.tu-berlin.de/~bbraatz/papers/BK06-TR.pdf`.

[EEPT06] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gariele Taentzer. *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer Science. Springer, 2006.

[GR04] Martin Große-Rhode. *Semantic Integration of Heterogeneous Software Specifications*. Monographs in Theoretical Computer Science. Springer, 2004.

Structural Patterns for the Transformation of Business Process Models*

Marion Murzek
Women's Postgraduate College for Internet Technologies (WIT),
Institute for Software Technology and Interactive Systems Vienna University of Technology, Austria
murzek@wit.tuwien.ac.at

I. Motivation and Problem

As companies discovered the benefits of Business Process Modeling (BPM), the use of Business Process (BP) models moved from a "luxury article" to an "everyday necessity" in the last years. Meanwhile many companies own thousands of models which describe their business. Since business changes over the years, e.g., business to business interoperability came up with new inventions in communication (Internet) and companies merge with others, there arises a need to keep existing business models up-to-date and to synchronize or translate them into a contemporary BPM language. To facilitate these scenarios, a model transformation technique for BP models is needed.

Current techniques or specifications used for defining model transformations, such as ATL [2] or QVT [5], operate at the level of metamodel elements. That means, that the definition of model transformation scenarios is based on the correspondences of the elements in two or sometimes more different metamodels. This concept covers the local correspondences of each element which is used for copying, splitting and merging elements. For transformation correspondences which require knowledge of the context in which an element is used, e.g. deleting elements or integrating new elements, these techniques offer additional imperative programming concepts which, however, lead to complex transformation definitions.

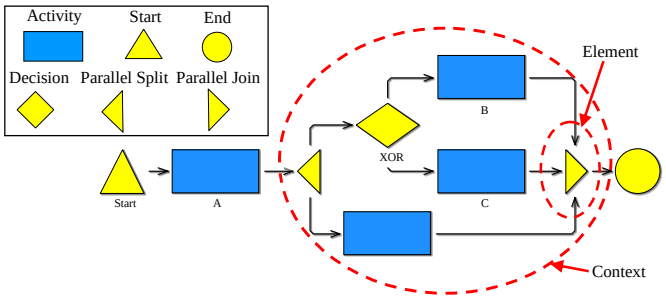

Fig. 1. Local view and context view

To reduce the complexity of defining context-dependent correspondences imperatively, we introduce structural patterns for the domain of BP models, as elaborated in [4]. These patterns make it possible to relate the elements of two BPM languages considering their context. So we can make use of the "bigger picture" as shown in Fig. 1.

*This research has partly been funded by the Austrian Federal Ministry for Education, Science, and Culture, and the European Social Fund (ESF) under grant 31.963/46-VII/9/2002.

II. Patterns to Simplify Transformation Definition

Patterns allow for a divide and conquer approach to model transformation. The transformation definition can be divided into *pattern matching*, i.e., analyzing the source model and identifying pattern occurrences, and *pattern instantiation*, i.e., synthesizing the target model such that the same patterns as in the source model appear. Defining a model transformation in two steps promises simplification of each step and re-use of steps when transformations between multiple BPM languages need to be defined.

As patterns encompass frequently occurring non-trivial BP model transformation requirements and their solutions, they are also a guidance for implementing such problems. So they could be seen as design patterns [3] for implementing model transformations for BP models.

Compared with for example the model transformation design patterns in [1] our patterns focus on the special requirements of model transformation in the area of BPM.

We presume that this approach copes with the fine-grained heterogeneities of BPM languages, because differences in particular details can be abstracted as higher-level patterns.

References

[1] J. Bézivin, F. Jouault, and J. Palies. Towards model transformation design patterns. In *Proceedings of the First European Workshop on Model Transformations (EWMT 2005)*, 2005.
[2] J. Bézivin, F. Jouault, and D. Touzet. An Introduction to the ATLAS Model Management Architecture. Technical Report 05-01, LINA, 2005.
[3] E. Gamma, R. Helm, and R. Johnson. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley, 1995.
[4] M. Murzek, G. Kramler, and E. Michlmayr. Structural Patterns for the Transformation of Business Process Models. In *Proceedings of the International Workshop on Models for Enterprise Computing (IWMEC 2006)*, October 2006.
[5] Object Management Group, Inc., http://www.omg.org/docs/ptc/05-11-01.pdf. *MOF QVT Final Adopted Specification*, November 2005.

Typed Polyadic π-calculus in Bigraphs*

Mikkel Bundgaard
The PLS Group
IT University of Copenhagen

Vladimiro Sassone
ECS
University of Southampton

3rd August 2006

Abstract

Bigraphs have been introduced with the aim to provide a topographical meta-model for mobile, distributed agents that can manipulate their own linkages and nested locations, generalising both characteristics of the π-calculus and the Mobile Ambients calculus.

In this presentation we briefly review recent work [1] on presenting type systems in bigraphs using sortings. In particular we examine the *typed polyadic π-calculus* with capability types of Pierce and Sangiorgi which we represent using a link sorting called *sub sorting*. We derive a labelled transition system which gives rise to a coinductive characterisation of a behavioural equivalence which is a congruence. The results obtained in [1] constitute a promising foundation for the presentation of various type systems for the (polyadic) π-calculus as sortings in the setting of bigraphs.

References

[1] Mikkel Bundgaard and Vladimiro Sassone. Typed polyadic pi-calculus in bigraphs. In *Proceedings of the 8th ACM SIGPLAN international conference on Principles and Practice of Declarative Programming (PPDP'06)*, pages 1–12. ACM Press, 2006.

*Supported by '**DisCo**: Semantic Foundations of Distributed Computation', EU IHP 'Marie Curie' HPMT-CT-2001-00290.

An Inductive Characterization of Matching in Binding Bigraphs*

Troels Christoffer Damgaard†
Joint work with Lars Birkedal†, Arne John Glenstrup†, and Robin Milner ‡

Bigraphs and bigraphical reactive systems, due to Milner and coworkers (see, e.g., Jensen and Milner [2004], Milner [2006]) have been proposed as a graphical model of mobile and distributed computation in which both locality and connectivity are prominent. In brief, a *bigraph* consists of a *place graph*, a forest, whose nodes represent a variety of computational objects; and a *link graph*, which is a hyper graph connecting ports of the nodes. Bigraphs can be reconfigured by means of *reaction rules*. Loosely speaking, a *bigraphical reactive system* (BRS) consists of set of bigraphs and a set of reaction rules, which can be used to reconfigure the set of bigraphs.

In the Bigraphical Programming Languages (BPL) research project at the IT University, we are working towards building a tool that will allow experimentation with BRSs. At the core problem of implementing the dynamics of bigraphical reactive systems is the *matching problem*, that is, to determine for a given bigraph and reaction rule whether and how a reaction rule can be applied to rewrite the bigraph. Essentially, we need to determine for an agent A, a context C, a redex R and a parameter d, when $A = CRd$ (where juxtaposition is composition). In this work we analyze that problem, and develop a sound and complete inductive characterization of matching of the so-called *binding* variant of bigraphs.

We work on *matching sentences* – defined as a 7-place relation among *wirings*, essentially link graphs, and *discrete* bigraphs, bigraphs with one-one global (unbound) wiring. Leaving out a few details, we write $\omega_{\mathbf{a}}, \omega_{\mathbf{R}}, \omega_{\mathbf{C}} \vdash a, R \hookrightarrow C, d$, for wirings $\omega_{\mathbf{a}}$, $\omega_{\mathbf{R}}$, $\omega_{\mathbf{C}}$, and a, R, C, d discrete bigraphs. We say that such a matching sentence, where $\omega_{\mathbf{R}}$ has global names Y, and d has global names Z, is *valid*, iff $(\mathsf{id} \otimes \omega_{\mathbf{a}})a = (\mathsf{id} \otimes \omega_{\mathbf{C}})(C \otimes \mathsf{id}_Y \otimes \mathsf{id}_Z)(\omega_{\mathbf{R}} \otimes \mathsf{id})(R \otimes \mathsf{id}_Z)d$, which (by a little rearranging) lets us capture the abstract definition of a match (as defined in Jensen and Milner [2004]).

We define an inference system consisting of nine quite simple rules, which suffice to infer all (and only) valid matching sentences. Conceptually, they each handle a separate aspect of bigraph. For example, a rule PAR explains how to handle (tensor) *product*, given two valid matches; LSUB allows us to match a bigraph with *bound* names by matching a bigraph with corresponding *unbound* names; and, a rule ION handles matching of *nodes*, by permitting us to find a match for an underlying bigraph with bound names (without the node).

Our results pave the way for a provably correct matching algorithm, as needed for an implementation of bigraphical reactive systems.

References

Lars Birkedal, Troels Christoffer Damgaard, Arne John Glenstrup, and Robin Milner. Matching of bigraphs. In *Proceedings of Graph Transformation for Verification and Concurrency Workshop 2006*, Electronic Notes in Theoretical Computer Science. Elsevier, August 2006.

Ole H. Jensen and Robin Milner. Bigraphs and mobile processes (revised). Technical Report 580, University of Cambridge, February 2004. ISSN 1476-2986.

Robin Milner. Pure bigraphs: structure and dynamics. *Inf. Comput.*, 204(1):60–122, 2006. ISSN 0890-5401.

*This work is under publication as Birkedal et al. [2006].

†IT University of Copenhagen, DK
‡University of Cambridge, UK