Electronic Communications of the EASST
Pre-proceedings

Proceedings of the
Eighth International Workshop on
Graph Transformation and Visual Modeling Techniques
(GT-VMT 2009)

Ten good reasons why structured graphs can be better than flat ones

Roberto Bruni

5 pages

Guest Editors: Artur Boronat, Reiko Heckel
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Ten good reasons why structured graphs can be better than flat ones

Roberto Bruni[1]

[1]bruni@di.unipi.it
Dipartimento di Informatica, Università di Pisa

Abstract: This talk presents our proposal, called ADR, for the design of reconfigurable software systems. ADR is based on hierarchical graphs with interfaces and it has been conceived in the attempt of reconciling software architectures and process calculi by means of graphical methods. We illustrate the main motivations behind ADR and the current advancements on its foundations, applications and tool support.

Keywords: Hierarchical graphs, ADR, Maude, graphical encoding

If you can find something everyone agrees on, it's wrong — M. Udall

1 Rationale

Graph-based notation, techniques and tools play a fundamental role for the formal analysis of complex systems. However, the complexity of modern software systems like those emerging in global computing and in service oriented architectures is such that the "representation distance" w.r.t. ordinary graph transformation methodologies has grown larger and larger and poses serious problems in terms of issues such as, e.g., scalability, open endedness, dynamicity or distribution. Furthermore, the same kind of complication hampers indiscriminately all phases involved in the modeling process (specification, design, validation and verification).

As old remedies should always be attempted first, we would like to show that one good way to tackle complexity and to enhance efficient formal reasoning is to superimpose some structure on complex graphs, whatever their nature is (e.g., Petri nets, BPEL-like workflow models, reference modeling languages, process calculi encodings). In other words, we propose to consider (a special kind of) structured graphs instead of flat graphs.

In the following we sketch some motivation to sustain our claim and then we shortly discuss our proposal called Architectural Design Rewriting (ADR). We assume the reader has some familiarity with the way in which typed (hyper)graphs are used to represent software architectures composed by components and connectors.

2 Ten Good Reasons

In this section we list some sample limitations of flat graphs. In all cases the use of structured graphs provides evident improvements.

2.1 Requirements

Type graphs are a convenient way to limit the way in which components can be connected together. However, there are some simple structural constraints that cannot be accounted for directly and need instead additional logic-based languages to be expressed properly. One such example is a pipeline architecture, where components are composed sequentially to form arbitrarily long sequences.

2.2 Model construction

Graph grammars, like those based on hyperedge replacement, can offer a convenient way to build properly shaped systems, but once the information about the generation process is lost then we are left with possibly large, flat, unstructured graphs that are not as representative of the modelled architectures as they could be. One such example is a tree-like architecture, where the information about the root is lost together with any logical dependency between components. A similar example is that of a ring architecture, where the components are generated sequentially by the rules of the grammar, but the information about leadership (the first generated component) is not encoded in the flat graph itself.

2.3 Parsing and browsing

Once a large graph has been constructed, it is important, for efficiency reasons, to have convenient ways to browse it, e.g. some sort of "blueprint" would help to identify and predicate over certain relevant subgraphs. One such example is a large loosely coupled network composed of local clique-shaped subnetworks, where characterizing such subnetworks is important for modeling software updates or load balancing.

2.4 Model conformance

Analogously, the "blueprint" offers an immediate witness that the undelying graph is conformant to a certain architectural style or to certain constraints required by the original specification. Recovering the analogous guarantee on the underlying graph from scratch is certainly more difficult.

2.5 Compositionality

The correct composition of software systems is often constrained by the way in which their interfaces can be connected. When flat graphs are considered, the information about their interfaces can be missing at all (i.e. suitable extensions need to be considered) or encoded ad hoc (but then possibly hard to recover).

2.6 Abstraction and refinement

The same system can be considered at different levels of abstraction, e.g. depending on the granularity of its components. For example, in the specification phase it can be the case that certain details are omitted in a first, rough modeling for fast prototyping but then they need to be inserted or reconsidered in a subsequent phase. If the graph is flat then it is more difficult to move from one level of abstraction to the other in a sound way. For example, consider a workflow of activities composed in series and in parallel at different degree of granularity: if the graph is flat then the grouping of subactivities must be encoded ad hoc to avoid confusion.

2.7 Graphical encoding

One important advantage of graphical encodings is the intuitive visual modeling of complex systems, where the graph itself should offer an immediate characterization of the represented system. This fact has been exploited with success, e.g., in the modeling of process calculi. In the case of process calculi for global computing and service oriented computing, there are some notions like sessions, transactions and nested compensations that require logical grouping of components or some notion of containment. For example, when a transaction is aborted it is important that every participant undertakes timely counteractions. Again, when flat graphs are considered, some ad hoc encoding of containment is necessary.

2.8 Evolution

Graph-based models are often equipped with graph rewrite rules in suitable formats that can account for complex reconfigurations by exploiting negative application conditions and sophisticated synchronization mechanisms. One big advantage is the availability of their solid theories related to concurrency and distribution. However, the graph matching problems arising for rule applicability are computationally more expensive than the pattern matching problems arising when a more structured approach is considered.

2.9 Reconfiguration

When reconfiguration rules are considered, like in self-repairing and self-adaptive systems, then it is important: 1) to guarantee the conformance of the reconfigured system; 2) to keep the reconfiguration as local as possible. In the case of flat graphs and ordinary graph transformation approaches these two features are often hard to match and need to be studied case by case. For example, consider the case of a global network that connect several star-shaped, local networks, whose servers are the components at the center of the star. Then, when a server goes down, a valid reconfiguration could consist of migrating all its peers to a different server, which is not easy to accomplish with ordinary SPO or DPO rules.

2.10 Logical specification and verification

Last but not least, the representation distance between the structural and behavioural properties of the modelled systems and the logical formulas needed to express them. Moreover, in the case of automatic verification it is crucial to limit state explosion and to guarantee efficient analysis. For example, when spatial logics are considered, then verification can be sensibly improved by exploiting some superimposed strucure over graphs.

3 Architectural Design Rewriting

Architectural Design Rewriting (ADR) is a proposal for the design of complex software systems. The key features of ADR are: 1) hierarchical graphs with interfaces; 2) algebraic presentation; 3) inductively-defined reconfigurations.

Roughly, the underlying model consists of some kind of interfaced graphs whose inner items represent the architectural units and their interconnections and whose interface expresses the overall type and its connection capabilites. Architectures are designed inductively by a set of productions which enable: top-down refinement, like replacing an abstract components with a possibly partial realisation; bottom-up typing, like inferring the type of an actual architecture; well-formed composition, like composing some well-typed actual architectures together so to guarantee that the result is still well-typed. In the functional reading, the set of productions defines an algebra of terms, each providing a proof of style conformance. Hence, the interpretation of a proof term is the actual architecture.

The dynamics is then expressed by term rewrite rules acting over proof terms rather than over actual architectures. This has many advantages, like guaranteeing that all reconfiguration are style-preserving by construction.

The flexibility of ADR has been validated over heterogeneous models such as network topologies, architectural styles and modelling languages. A prototypical implementation of ADR has also being developed using Maude, together with a simple visualisator of term-like specifications of hierarchical graphs. The web site of ADR (http://www.albertolluch.com/research/adr) makes both tools available and provides links to several additional resources.

Acknowledgements: The work on ADR has been done in collaboration with several colleagues, with whom I have been honoured to exchange many ideas, compare different perspectives and approaches, share stress and friendship and whose contribution has been fundamental for the topics of this talk: my best thanks for that to Alberto Lluch Lafuente, Ugo Montanari, Emilio Tuosto, Fabio Gadducci, Antonio Bucchiarone, Stefania Gnesi and Dan Hirsch.

I would also like to thank Artur Boronat and Reiko Heckel for inviting me to give this talk at GT-VMT 2009.

Research on ADR is supported by EU FET-GC2 IP project SENSORIA, IST-2005-016004.

Bibliography

[BBG+08] R. Bruni, A. Bucchiarone, S. Gnesi, D. Hirsch, A. Lluch Lafuente. Graph-Based Design and Analysis of Dynamic Software Architectures. In Degano et al. (eds.), *Concurrency, Graphs and Models, Essays Dedicated to Ugo Montanari on the Occasion of His 65th Birthday*. Lect. Notes in Comput. Sci. 5065, pp. 37–56. Springer Verlag, 2008.

[BGL09] R. Bruni, F. Gadducci, A. Lluch Lafuente. Graphical Representation of Process Calculi via an Algebra of Hierarchical Graphs. 2009. Manuscript submitted to a conference. Available at http://www.albertolluch.com/papers/adr.algebra.pdf.

[BLM09] R. Bruni, A. Lluch Lafuente, U. Montanari. Hierarchical Design Rewriting with Maude. In Rosu (ed.), *Proceedings of WRLA 2008, 7th International Workshop on Rewriting Logic and its Applications*. Elect. Notes in Th. Comput. Sci. Elsevier Science, 2009. To appear.

[BLMT08a] R. Bruni, A. Lluch Lafuente, U. Montanari, E. Tuosto. Architectural Design Rewriting as an Architecture Description Language (position paper). In Bhargavan et al. (eds.), *Proceedings of R2D2 2008, Workshop on the Rise and Rise of Declarative Datacentre*. Pp. 15–16. 2008. Technical Report MSR-TR-2008-61, Microsoft Research Cambridge.

[BLMT08b] R. Bruni, A. Lluch Lafuente, U. Montanari, E. Tuosto. Service Oriented Architectural Design. In Barthe and Fournet (eds.), *Proceedings of TGC 2007, 3rd International Symposium on Trustworthy Global Computing*. Lect. Notes in Comput. Sci. 4912, pp. 186–203. Springer Verlag, 2008.

[BLMT08c] R. Bruni, A. Lluch Lafuente, U. Montanari, E. Tuosto. Style-Based Architectural Reconfigurations. *Bulletin of the EATCS* 94:161–180, 2008.