Dynamic Accommodation of Change: Automated Architecture Configuration of Distributed Systems

Gabriele Taentzer
Computer Science Deptartment
Technical University of Berlin, D - 10587 Berlin
gabi@cs.tu-berlin.de

Michael Goedicke and Torsten Meyer
Dept. of Mathematics and Computer Science
University of Essen, D - 45117 Essen
{goedicke,tmeyer}@informatik.uni-essen.de

Abstract

A major challenge in developing and coordinating distributed agents is to accommodate changes introduced at one agent and to propagate it to every other interested party. This presents a special difficulty if the information maintained in a node is highly structured. Examples of such problems are dynamic accommodation of structural changes in distributed software systems or consistency management of multiple viewpoints in typical multiple perspectives/stakeholders settings: modifications or changes not envisaged at "design time" have to be handled at run time without disturbing those parts of the system unaffected by the change. The key feature realizing such dynamic modifications and extensions is a clean separation between internal operations or actions of an individual node or agent and their structural coordination. Thus, general change rules for creation/deletion and connection/disconnection of nodes/agents can be formulated. In this contribution we consider as an important problem area dynamic change management of distributed systems. We propose to use distributed graph transformation as the underlying formalism to realize both the specification of evolving distributed systems as well as dynamic change management, thus taking a step towards building configurable distributed systems. The results obtained can easily be transferred to the field of consistency management.

1. Introduction and related work

Change in distributed systems has at least two aspects. On the one side there is the problem to maintain a certain - application dependent - notion of consistency among the distributed components without having access to the entire state of the distributed system all the time. On the other side there is the general need to evolve a distributed system at the structural level by adding, removing or substituting software components during the runtime of the system.

Thus dynamic coordination of change is inevitable and systematic management of inconsistencies is highly desirable as well. While managing (in)consistency in multiple ViewPoints [3] is directly related work, in this contribution we consider as a special case the problem to describe the systematic change of a distributed system's architecture. Thus we use this domain to show how dynamic change of complex structures can be accommodated systematically.

In the case of distributed systems, graph transformation can be employed twice:

1. to describe the dynamic reconfiguration of distributed systems, and
2. to model evolving data and object structures in the local systems.

In designing configurable distributed systems the two levels mentioned above occur in a natural way. The design involves the architecture of the system which means to find the overall structure and the elements of the structure. In a distributed software system certain mechanisms need to be implemented in order to achieve some - special - notion of consistency, since a global state is not accessible all the time. In the case of dynamic (re)-configuration of such systems additional caution has to be exercised that exchange, introduction and removal of system components is done in a smooth way without disturbing the rest of the system too much. In general such an architectural change implies the setting of involved components into a kind of silent state such that the architectural change actions can be performed and the new parts may be activated in turn.

Our approach of distributed system design by distributed graph transformation allows to export parts of the internal structures as well as dynamic behaviour which means to export also parts of the semantics. This information is extremely useful to find, for example, adequate services on remote systems, since syntactic information like names or signatures are often not expressible enough to find the right service. Another advantage is the possibility to dynamically change interfaces which takes care of the fact that not all services can be offered all the time. This property can be described in a natural way using our approach since interfaces are again defined by graphs and graph rules.

Prominent work related to configurable distributed systems is done at Imperial College, London. In [6, 5] J. Kramer and J. Magee propose a basic model for dynamic change management which separates structural issues from application specific ones, thus distinguishing change rules at the architecture configuration level and change actions at

the component level. In [7] self-configuring architectures are investigated.

An approach to realize configurable distributed systems based on negotiation of component semantics by the help of graph transformation techniques is presented in [2]. First ideas of the approach presented in this paper are formulated in [10]. Also in [10] the advantages of our approach are demonstrated by presenting a ring database example which has formerly been modeled with the approach of J. Kramer and J. Magee in the architecture description language DARWIN/REGIS [5].

The concepts as well as the formal definition of distributed graph transformation based on the algebraic approach [1] are introduced in [8]. Using graph transformation as underlying formal framework, consistency of the resulting distributed system structures is ensured.

In the following we will give a brief account of dynamic change management in distributed systems in chapter 2. In chapter 3 we will present distributed graph transformation as the technique to realize these goals. This chapter will also sketch the use of distributed graph transformation to realize dynamic change management with the final aim of configurable distributed systems.

2. Dynamic change management and system configuration

As mentioned above change in distributed systems has to be managed at the structural / configuration level as well as at the component level. We follow the change model developed in [6, 5]. This model implies a strict separation of the two change aspects.

Separation of concerns applied to the design of distributed systems implies the separation of designing the various components from designing the structure of the system. This is naturally transferred to the management of change in such systems. The change of the distributed system structure has to be done at the level of the structure and the node interfaces only. This implies that the interaction between the change actions and the nodes of system is minimal. As a consequence the evolution of the system can be performed in a structured and systematic way.

The idea outlined above requires a certain view on the system structure and its nodes. The nodes offer an interface which provides their publicly available services which are linked to each other according to the system structure via communication channels.

The actions at the structural level - which we will refer to as the network level later on - are *addition* and / or *deletion* of a node or subsystem and *linking* and / or *unlinking* of nodes. This is the original model of change actions according to [6]. It minimizes the interaction between the change actions at the network level and the related actions necessary within the involved nodes. I.e. involved nodes have to be in a *quiescent* state in order to allow the performance of the change action at network level. A node is said to be in a quiescent state if the application's state is consistent and there is no communication in progress between nodes affected by a change nor between them and their environment. This means that a distributed node is in a quiescent state if:

1. it is not currently engaged in a communication connection that it initiated,
2. it will not initiate new communication connections,
3. it is not currently engaged in servicing a communication connection request,
4. no communication connections have been or will be initiated by other nodes which require service from this node.

If only the first requirement is satisfied a node is said to be in a *passive* state.

Further, the nodes must remain quiescent while the entire change is executed. Thus the only interaction between the structural change actions and the nodes are services which activate and / or passivate a node.

This view, however, has a certain drawback. It does not deal with the situation that a node is split into more than one node. Also in the case of adding a new node there is the problem of informing the new node about the state it has to know in order to co-operate smoothly with the existing nodes of the system. Thus we would like to extend the model by means to describe the interaction between nodes more in detail. Still the interaction has to be done through interfaces which hide internal details from the (mis)use by other nodes. Our approach provides the possibility for multiple interfaces which allow to deal with different aspects using specialized interfaces. Below we now introduce the technique used to realize dynamic change management: *distributed graph transformation*.

3. Distributed graph transformation and its integration into the change model

Graph transformation means the rule-based manipulation of graphs, e.g. *if* a certain structure exists *then* it may be transformed into a new one. In system modeling, graphs are used to describe states while rules describe state transitions. There are several approaches to graph transformation. We follow the *double-pushout approach* as presented in [1] which not only contains an introduction to the approach, but also an overview on the important theoretical results achieved so far. In this approach, each node and edge of a graph may be typed and attributed by further data types. A *rule* consists of three graphs, the *left* and the *right-hand sides* as well as a *gluing graph*. The gluing graph is part of the other two graphs and contains all those nodes and arcs which should be preserved during a transformation. All graph objects which are additionally in the left-hand side are to be deleted while all graph objects additionally in the right-hand side are to be newly created. The gluing graph is useful to specify where to insert the newly

created graph objects. An application of a rule to a graph is performed by first finding a *match* of the left-hand side into the graph and then, applying the rule to the graph at this match. A start graph which describes the initial state together with a set of rules specifying the actions allowed form a *graph transformation system*.

Distributed graph transformation as presented in [8] distinguishes two levels of abstraction: the *network* and the *object (or node)* level. The network level contains the description of a system's network topology by a network graph and its dynamic reconfiguration during runtime by network rule applications. Each network node may contain the description of a local system, again described by a graph transformation system. At this abstraction level, graphs describe object structures and rules are used to model object interactions.

The combination of the network graph structure and the local object structures is specified by distributed graphs in order to describe distributed object structures. A *distributed graph* consists of one *network graph* where each network node is refined by a local graph describing local object structures. Each network edge is refined by a mapping on local graphs defining which graph parts are in im- or export interfaces or replicated from others. In export interfaces, local systems present object structures accessible for other local systems and import interfaces contain object structures required from remote export interfaces. Communication between local systems only takes place via export and import interfaces. Usually, each local system contains different sets of node, edge and attribute types. The interfaces may contain any subsets of types for communication (cf. [8] for more details on distributed graph attribution). Figure 1 shows an example of a distributed graph modeling three database schemes in a ring (cf. [10, 5] for a detailed elaboration of the ring database example).

A distributed graph rule consists of a network rule and object rules for all network nodes which are preserved by the network rule. An example of a distributed rule is given in Figure 2 showing a change management rule for removing a database. Distributed graph rules are useful to describe dynamic network reconfiguration by a network rule as well as dynamic local data structures in each component by a set of local rules. Furthermore, remote interactions can be described by distributed rules. In this case, all network nodes where parts of the interaction should take place, are incorporated into the network graphs of the rule.

A distributed graph transformation may change both the network structure as well as object structures in local systems and their interfaces. For performing these actions first all distributed matches have to be found. If all matches are fixed there always exists a unique result for a distributed graph transformation (see [8]).

The advantage of using distributed graph transformation in this application field is the possibility to describe the network as well as the object structures visually. The rules are useful to avoid the need to define a global control flow. Each action is modeled by one or more rules and nothing is said how the rule applications interact. This depends of the rule contents only.

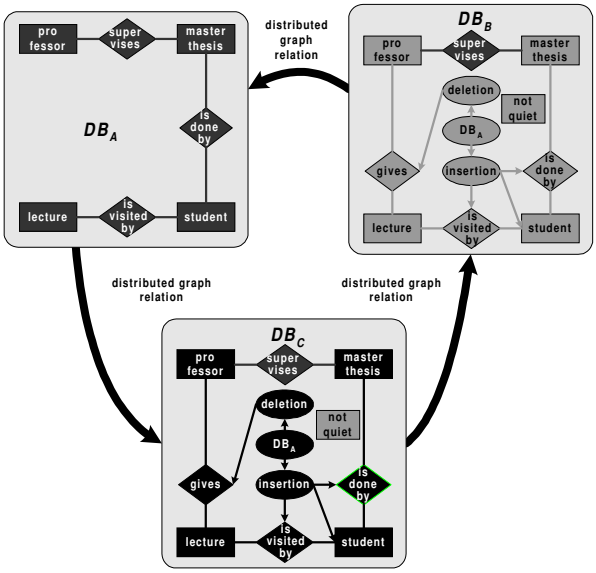

Figure 1 An update has been completed in database DB_A and is exported by DB_B and imported in DB_C (assuming ER diagrams as the underlyig data structure). Body elements of local graphs are depicted in ***medium gray***, exported graph elements are depicted in *light gray* and imported graph elements are depicted in ***black***. Wrt. the distributed graph relation a mapping between graph elements with identical labels is assumed. Please note that in this context, the node passive state is equivalent to node quiescence, because we assume synchronous communication channels which do not lose any information [5].

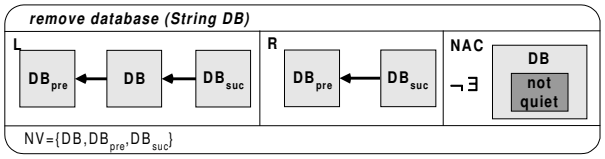

Figure 2 This distributed rule represents a change management rule for dynamically removing databases. Node label variables are given by the set NV and a negative application condition (NAC) is the non-existence of a 'not quiet' flag in database DB (cf. [6] for quiescence).

Distributed graph transformation is a natural formalism to realize the change model presented in chapter 2. While application-internal states and behaviour can be described by distributed graph transformation at the local level, distributed graph transformation at the network level is well suited for dynamic change management activities [10].

Distributed graph transformation satisfies the requirements to dynamic change management postulated in [6]:

- *Changes are specified in terms of the system structure:* this is achieved by formulating the structural change rules as network level graph transformation rules.

- *Change specifications are declarative:* this is achieved by the graph transformation approach which uses rules to define a transformation. With graph rewrite rules, application order does not have to be specified explicitly.
- *Change specifications are independent of algorithms, protocols, and states of the application:* this is achieved by separating network and node level of the distributed system. The approach to distributed graph transformation ensures the necessary conditions for consistent interaction.
- *Changes leave the system in a consistent state:* this is achieved by formulating the transformation rules in the way the notion of system consistency requires.
- *Changes minimize the disruption to the application system:* this is achieved by the rule-based approach; i.e. the left-hand side of a rule precisely delineates the involved area.

4. Conclusion and further work

In this contribution we have introduced our approach to distributed graph transformation applied to the problem of dynamic change management in distributed systems. Distributed graph transformation nicely satisfies all the requirements postulated in [6]. Moreover, by using graph transformation coherently at both levels - the network level and the node level - more detailed and specialized change actions can be described. This is done while the encapsulation provided by the nodes is preserved using only the node's interfaces. In [10] we have shown a sample system where application specification and change management rules as well as the interaction between both levels can be realized by distributed graph transformation thus taking a step towards building configurable distributed systems.

However, if it is not appropriate to specify the entire distributed system by graphical means, our approach allows to combine it with a general architectural system description in terms of distributed software components and interconnections. This related work is pursued in [2]: here semantic properties of components are specified graphically and the negotiation aspect in multiple client/server relations is emphasized.

As stated in chapter 2, consistency is a very important aspect in dynamic change management. A notion of consistency conditions, logical formulas on the existence and non-existence of graph structures, and a constructive approach to ensure these conditions are presented in [4]. It is due to future work to extend this work to distributed graph transformation.

We will use the results obtained here to describe the problem of managing the potentially inconsistent information contained in multiple ViewPoints [3]. The two level of graph representation introduced here are well suited for representing multiple View-Points and their relations. The various sorts of consistency checks are then expressed using distributed graph transformations in a natural and systematic way.

Currently we are working on an implementation of our ideas which will be based on the current AGG - machine [9]. Of course, the implementation will also make use of a distributed computing system. In a first approach, we plan to support the definition of import and export interfaces for graphs and rules realized as substructures of those. The main attention lies on the synchronization of local transformations related by subtransformations. The communication will be realized by remote method invocations (RMI) supported by Java.

5. References

[1] Corradini, A., Montanari, U., Rossi, F., Ehrig, H., Heckel, R., Löwe, M., "Algebraic Approaches to Graph Transformation", in Rozenberg, G. (ed.), *Handbook of Graph Grammars and Computing by Graph Transformation Vol. 1 Foundations*, pp 163 - 245, World Scientific, Singapore, 1997.

[2] Goedicke, M. and Meyer, T., "Dynamic Semantics Negotiation in Distributed and Evlving CORBA Systems: Towards Semantics-Directed System Configuration", *Proc. 4th ICCDS*, Annapolis, USA, IEEE Press, 1998.

[3] Goedicke, M., Meyer, T., and Taentzer, G., "ViewPoint-oriented Software Development by Distributed Graph Transformation: Towards a Basis for Living with Inconsistencies", *Proc. 4th IEEE ISRE*, Limerick, Ireland, 1999.

[4] Heckel, R. and Wagner, A., "Ensuring Consistency of Conditional Graph Grammars -- A constructive Approach", *Proc. SEGRAGR'95*, Electronic Notes of TCS, Vol. 2, http://www.elsevier.nl/locate/entcs/volume2.html, 1995.

[5] Kramer, J. and Magee, J., "Analysing Dynamic Change in Software Architectures: A Case Study", *Proc. 4th ICCDS*, Annapolis, USA, IEEE Press, 1998.

[6] Kramer, J. and Magee, J., "The Evolving Philosophers Problem: Dynamic Change Management", *IEEE Transactions on Software Engineering*, SE-16, 11, pp. 1293-1306, 1990.

[7] Magee, J. and Kramer, J., "Self Organizing Software Architectures", *Proc. SIGSOFT Workshops*, ACM Press, 1996.

[8] Taentzer, G., Fischer, I., Koch, M., and Volle, V., "Distributed Graph Transformation with Application to Visual Design of Distributed Systems", to appear as article in Rozenberg, G. (ed.), *Handbook of Graph Grammars and Computing by Graph Transformation Vol. 3 Concurrency and Distribution*, World Scientific, Singapore, 1999.

[9] Taentzer, G., Ermel, C., and Rudolf, C., "AGG-Approach: Language and Tool Environment", to appear as article in Rozenberg, G. (ed.), *Handbook of Graph Grammars and Computing by Graph Transformation Vol. 2 Specification and Programming*, World Scientific, Singapore, 1999.

[10] Taentzer, G., Goedicke, M., and Meyer, T., "Dynamic Change Management by Distributed Graph Transformation: Towards Configurable Distributed Systems", 6th *International Workshop on Theory and Application of Graph Transformations*, Paderborn, Germany, 1998.