Model Driven Capabilities of the DA-GRS Model

Arnaud Casteigts

Model Driven capabilities of the DA-GRS model

Arnaud Casteigts
LaBRI, Université Bordeaux 1,
351 Cours de la Libération, F-33405 Talence, France.
E-mail: arnaud.casteigts@labri.fr
http://www.labri.fr/

***Abstract*— The development of applications that target dynamic networks often adresses the same difficulties. Since the underlying network topology is unstable, the application has to handle it in a context-reactive manner, and, when possible, with algorithms that are localized and decentralized. The present paper shows how the DA-GRS model, that is a high level abstract model for localized graph algorithms, can be used to drastically improve the development time of self-organized systems. The canonical method that is proposed makes use of generic pieces of code, along with some additional code generation. All the concepts presented here are illustrated by means of a spanning forest algorithm, that can then be used by the application at a high abstraction level.**

***Index Terms*— context-awareness, localized algorithms, dynamic graphs, self-adaptability, design methods for self-organizing systems, mobile computing.**

The work presented in this paper is carried out at LaBRI (Laboratoire Bordelais de Recherche en Informatique) and more precisely in the SOD (Distributed Systems and Objects) team. It is partly achieved within the framework of the Sarah (Asynchronous services for mobile ad-hoc networks) project supported by the ANR (Agence Nationale de la Recherche).

I. Introduction

Pervasive services and mobile computing have raised many new possibilities. The ultimate goal is the ability to access a given technology anywhere, anytime, in a seamless manner. While major improvements have been done in the area of devices themselves, with more and more cpu, memory capabilities, and communication technologies, one of the main challenges remains the design of software layers adapted to this new context.

Such environments require various features they must be provided with. First, a device must be aware of its context. This includes at least mechanisms that make it possible for a node to react to the appearance or disappearance of direct neighbors, whatever the reason. Context-awareness can also go further, by taking into account a lot of other parameters, such as the density, the signal strengh, the kind of neighbor devices, etc.

Second, when the network is not infrastructured (*i.e.* ad-hoc), the devices have to self-organize and set up local group communication systems, from a simple one-hop connection to optimized overlay structures such as spanning trees.

Third, in a pervasive environment where communication links can break at anytime, distributed algorithms have to be localized, *i.e.* a device should interact with direct neighbors only. This matches the reality and makes it possible, among other things, to react to an event in one single computation step.

The DA-GRS model (Dynamicity-aware Graph Relabeling System) is a very high level computation model abstraction based on graphs. The two main properties of this model are localization and dynamicity, *i.e.* the model supports the design of algorithms in which a computation step involves only direct neighbors (or k-hops neighbors), and where a device (represented by a vertex) can react to the appearance/disappearance of its neighbors.

The rest of this paper is organized as follows. Section II presents the DA-GRS model, and explains why it is well adapted to model self-organized systems and context-aware computing. In section III we describe the model driven capabilities that can be plugged in the DA-GRS model and show how these capabilities, added to generic pieces of software, can drastically reduce development efforts and make it possible for developers to focus on the application level. All these concepts and ideas are illustrated in section IV by means of the development of an application skeleton that sits on top of a spanning tree topology in a mobile ad-hoc network. We eventually conclude and discuss perspectives in section V.

II. The DA-GRS model

The DA-GRS model has been introduced in [1] as a high level abstraction model that helps in designing and simulating distributed algorithms in a dynamic context. This model is an adaptation of the Graph Relabeling Systems (described in [2]), to the paradigm of dynamic and self-organizing networks. The main characteristics of the DA-GRS model, that are locality and dynamicity, make it a suitable tool to represent the core mechanisms that an application has to deal with in order to handle an unpredictable changing context.

A. The network

The network is represented by an undirected loop-less dynamic graph $G = (V, E)$, with V being the set of vertices representing the mobile units (or nodes) and E being the set of edges such that: $\forall x, y \in V, (x, y) \in E \iff x$ and y can communicate directly.

The dynamicity of the network is represented by the fact that V and E can change anytime with the following meaning:

- A vertex v is added to (resp. deleted from) V if the corresponding mobile unit is turned *on* (resp. *off*). Note that the deletion of v is equivalent, from a communication point of view, to the deletion of all the edges incident to v in one step.
- Let $dist(v1, v2)$ be any distance function between two nodes (usually the euclidean distance) such that $dist(v1, v2) < threshold \iff v1$ and $v1$ can communicate. Then an edge $e = (v1, v2)$ is added to E if and only if $dist(v1, v2) < threshold$ and $e \notin E$. Symmetrically, an edge $e = (v1, v2)$ is deleted from E iff $dist(v1, v2) > threshold$ and $e \in E$.

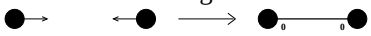
B. Labeling

The state of nodes and communication links are coded by means of vertex and edge labels. Each vertex has a state label for itself and another state label for each of its incident edges. An edge thus has a (possibly different) label on each side, which permits to code a non-symmetrical state (e.g. its orientation).

When an edge is added to the graph, it has an initial default label (noted 0). When an edge is deleted, its endpoint nodes add a special label to code the fact that the communication link has broken. This special label, noted off will allow (if desired) to apply some special operation to handle the deletion of the edge; thereafter, the edge is definitely and locally deleted.

The operations of addition and deletion of edges can thus be represented as in Fig. 1.

- Addition of an edge:



- Deletion of an edge:

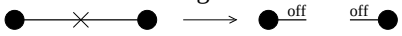


Fig. 1. Addition and deletion of edges, according to topological changes

C. Computation

To provide the highest level of abstraction as possible, still matching the reality of a distributed system, the DA-GRS model considers a computation step as a relabeling rule involving two direct neighbors. According to the states of the two vertices along with the state of their common edge, the rule computes a new value for all the involved labels. Let us consider the example of an algorithm that propagates a piece of information in a network. The semantics attached to the label is as follows:

- the value A (resp. N) for the label of a vertex means that the corresponding node has (resp. does not have) the piece of information.
- the value 1 (resp. 2) for the endpoint label of an edge, means that the message has been sent (resp. received) using this edge.

The entire algorithm can then be coded by the simple relabeling rule shown in Fig. 2.

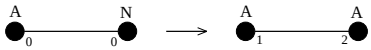


Fig. 2. The propagation algorithm example

D. Abstraction

The communication models that are usually considered in distributed systems are message passing, mail boxes or shared memory, some of which can be synchronous or asynchronous. The choice of a given model is decisive and often implies different approaches for the same problem. With the DA-GRS approach, the communication model is abstracted. A computation step is defined by a pair (pre-state, post-state), which makes it possible to focus on a general solution for a given problem. The way a GRS based algorithm can be adapted to a specific communication model (and especially to the asynchronous message passing model) has been discussed many times, see for instance [3] or [4] for more information.

Concerning the synchronization of nodes, i.e. the way nodes choose each other to compute one step, there exist several solutions that have different advantages and disadvantages. Here are three synchronization layers that can be used by DA-GRS:

- The rendezvous [5] is a synchronization mode in which, at each step, each node chooses one of its neighbors at random. If two neighbors have chosen each other, then there is a rendezvous between them and a computation step between these nodes can be performed. The advantage of this synchronization mode is that it is fair and non-deterministic (it allows to study the impact of network topologies on the execution of a given distributed algorithm). The disadvantage is that it is not efficient. An analysis of the rendezvous synchronization has been done in the context of static graphs and can be found in [6].
- The constraint based synchronization has been introduced in [7]. This synchronization mode does not care about fairness, and allows to specify criterions based on the communication links, in order to favour interactions between neighbors that match it. Such criterions are for instance the latency, the average bitrate, the signal strength, etc.
- An hybrid synchronization mode is currently being studied, in which neighbors select each other at pseudo-random, according to probabilities that match the same kind of criterions as for the constraint based synchronization.

Since the present paper focuses on model driven features rather than performance issues, we will abstract this layer by considering it as a way to select a pair of neighbors for a computation step, as shown in Fig. 3.

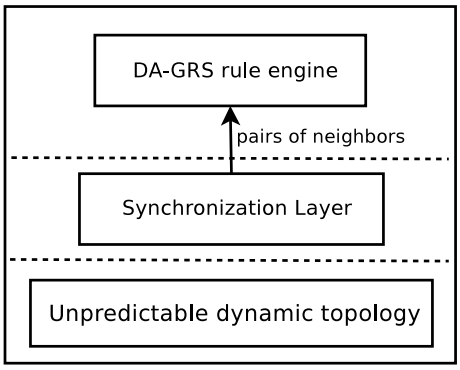


Fig. 3. Abstraction of the synchronization layer

III. Model driven development

The core mechanisms on which self-organized systems rest, often correspond to classical organization schemes such as spanning trees, covering rings, *k*-connected spanning structures, etc. These are precisely the kind of structures that are easily modeled by DA-GRS. The localized and dynamic properties of the model make it possible to develop distributed algorithms that are by nature able to (self-)organize a network in a context-reactive manner. This section presents a canonical method to develop an application that uses a DA-GRS based algorithm to manage its topological organization. Examples of code and object concepts are inspired from the Java language.

A. The core algorithm

According to the desired organization scheme, the developer has to design the appropriate DA-GRS algorithm $A = (I, R)$ with I standing for the set of initial labels and R for the set of relabeling rules.

B. Execution of the algorithm

Thanks to the rule engine embedded on each node (that synchronize each other using one of the synchronization mode presented in section II), the runtime is able to execute a DA-GRS algorithm. Once again, the execution of such an algorithm is beyond the scope of this paper that focuses on the use of these tools by an application developer. An embedded DA-GRS rule engine has been developed for a grid of PCs [8] and is currently being ported to a number of smartphones that use J2ME and Bluetooth. A simulator that uses a similar rule engine has also been developed and is freely available at [9].

C. Integration

The algorithm being just sets of states and rules, the goal is to make it possible for the application to give it a sens at its own level, without having to consider the rule engine internal working.

1) Which kind of integration: When the rule engine is in the process of applying a rule with another node, it should inform the application, that will then execute the appropriate code to handle that, according to what it means at its level. The system we propose is based on an upcall mechanism that takes place through an automatically generated interface.

2) Interface generation: Based on a given algorithm $A = (I, R)$ an interface is generated. This interface specifies the methods that have to be implemented (one for each rule). Parameters that are passed through these calls are a reference to an object that represents the considered neighbor, along with a boolean that codes which side of

the rule the underlying node it sitting at (from an operational point of view, the rules are not symmetrical).

Let us consider $R = \{r1, r2\}$. The generated interface will look as shown in Fig. 4.

```
public interface MyAlgoListener{
  public void onR1(Node, boolean);
  public void onR2(Node, boolean);
}
```

Fig. 4. Example interface

3) Use by an application class: To use the whole framework, the developer only has to provide a class that implements the generated interface, and to write the code that implements the operational aspect of the algorithm in the appropriate methods. Fig. 5 shows an implementation for the example interface of Fig. 4.

```
public class MyClass
             implements MyAlgoListener{
  DaGRS re; // rule engine creation
  MyClass{
    // instanciation of the rule engine
    re = new DaGRS (''MyAlgo.dagrs'');
    // registration for upcalls
    re.setListener (this);
  }
  public void onR1(Node n, boolean side){
    // code for the semantics of rule 1
  }
  public void onR2(Node n, boolean side){
    // code for the semantics of rule 2
  }
}
```

Fig. 5. Use example

Note that upcalls will take place precisely before the assignment of the new states in the rule engine, and that the supplied code will be executed on both sides at the same time. If the communication link is broken during the execution of the application level code, the architecture should provide a way to propagate and then to handle that at the same application level. This could be achieved by raising an exception, that the developer would catch in its implementation.

IV. Illustration

This section illustrates the use of the DA-GRS model driven capabilities by means of the example of an application that sits on top of a spanning tree in a dynamic network. All code examples and object concepts are inspired from the Java language.

A. The spanning tree algorithm

A spanning tree of a graph $G = (V, E)$ is a connected cycle-free subgraph $G' = (V', E')$ of G such that $\forall v \in V, v \in V'$. In the context of a dynamic network, it would be more precise to talk about a spanning forest, since the network can be partitionned. The algorithm that is presented in this paper, $A = (I, R)$, guarantees to maintain anytime a spanning forest that strives for a spanning tree, using only one-hop context information (*i.e.* it is a purely localized algorithm).

Initially, each node is labeled J, *i.e.* $I = \{J\}$. The algorithm is composed of four rules, *i.e.* $R = \{r1, r2, r3, r4\}$.

Two representations can be used for DA-GRS algorithms. The first, and most intuitive, is the visual representation based on graphs, as shown in Fig. 6. The second, that is the one used by the DA-GRS rule engine, and that is equivalent, is shown Fig. 7. In this representation, each rule is coded by two statements: the first corresponds to the left side, *i.e.* pre-state, of a rule; the second corresponds to the right side, *i.e.* the post-state of the rule.

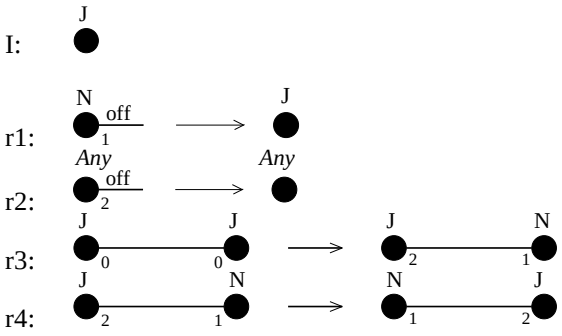


Fig. 6. Spanning forest algorithm (visual representation)

```
label,J  // initial state
=
// R1
v1.edgestate = off & v1.edgelabel = 1
v1.label = J & v1.edgelabel = 0

// R2
v1.edgestate = off & v1.edgelabel = 2
v1.edgelabel = 0 // allows the edge
                 // to be locally deleted

// R3
v1.label = J & v2.label = J
v1.edgelabel = 2 & v2.edgelabel = 1 \
                 & v2.label = N

// R4
v1.label = J & v2.label = N & v1.edgelabel ! 0
v1.label = N & v2.label = J & v1.edgelabel = 1 \
                            & v2.edgelabel = 2
```

Fig. 7. Spanning forest algorithm, as used by the rule engine

B. Explanation of the algorithm

The algorithm is based on three operations on a token: circulation, merging and regeneration. Ini-

```
public interface SpanningForestListener{
  public void onR1(Node);
  public void onR2(Node);
  public void onR3(Node, boolean);
  public void onR4(Node, boolean);
}
```

Fig. 8. The SpanningForestListener interface

tially, each node has a token (and is labeled J), meaning that each node is a spanning tree in itself, containing exactly one element (itself), and being its own root.

When two nodes labeled J meet each other, thanks to $r3$, the two spanning trees merge. Indeed, the labels 1 and 2 on an edge mean that it is part of the spanning tree. The use of two different labels allows a node to know the local route to the token. When $r3$ applies, one of the two tokens is deleted and one of the nodes is relabeled N, that guarantees that there is at most one token per tree.

The rule $r4$ codes the circulation of the token in a tree of the forest. Note that the edge labels are switched to ensure that the local route to the token remains consistent.

When a communication link is broken, *i.e.* when an edge is deleted, the node that is on the token side has nothing to do regarding the token maintainance, and simply applies rule $r2$. The node that had the deleted edge label to 1 has lost the route to the token, and is the only one of its remaining piece of tree to know that. It then regenerates a new token thanks to rule $r1$.

C. Priorities

The DA-GRS model considers light priorities between rules in the sense where once a neighbor is chosen, the rules are tried in the algorithm order until one can be effectively applied. Moreover, the disappearance rules (e.g. $r1$ and $r2$) have a higher priority than the other rules and do not need a neighbor selection in order to be applied. Note that according to these priorities, a node that has to regenerate a token cannot be used for anything before having effectively regenerated the token. As a side effect, this ensures that there is one and only one token in the tree, at anytime.

D. Interface generation

The generated interface is thus as shown on Fig. 8. Note that for the disappearance rules, a *boolean side* parameter would be useless. The *Node* parameter is here just a reference to the object that represented the disappeared neighbor.

```
public class MyClass
       implements SpanningForestListener{
  private DaGRS re; // rule engine creation

  public Vector neighbors;
  public Node father;
  public Node me;

  MyClass{
    re = new DaGRS (''spanforest.dagrs'');
    re.setListener (this);

    neighbors = new Vector();
  }
  public void onR1(Node n){
    neighbors.remove(n);
    father = null;
  }
  public void onR2(Node n){
    neighbors.remove(n);
  }
  public void onR3(Node n, boolean side){
    neighbors.add(n);
    if (!side){
      father = n;
    }
  }
  public void onR4(Node n, boolean side){
    if (side){
      father = n;
    }else{
      father = null;
    }
  }
}
```

Fig. 9. Implementation of the spanning forest meaning

E. Use of the spanning forest in an application

The spanning forest can then be used in a seamless way by an application, provided that the developer has added the code that gives sens to the algorithm in the corresponding methods. Fig. 9 proposes a very basic code that will allow the remaining of the application to use the properties of a dynamic spanning tree in a seamless manner.

F. Comments about the algorithm

In the code example of Fig. 9, the term of *father* has been used to identify the route to the token. This use is witting, in the purpose of showing that the token can be considered as a moving tree root, since it is unique in each tree.

An interesting point is that the DA-GRS algorithm presented here has been used as a spanning forest structure maintenance tool. With the same set of rules it could be used with a different high level code, and thus meaning, for example to elect a leader (the owner of the token) or to control an exclusive access to a communication medium.

G. Simulation

All DA-GRS algorithms, provided that they are expressed in the pseudo-logic formalism of Fig. 7

can be simulated and visualized in the DA-GRS simulator [9]

V. Conclusion

The domain of pervasive services and mobile computing is a research area that is still emerging and where a lot of work remains to be done. The present paper has introduced the first step to generate self-organized and dynamic systems based on a graph relabeling approach, using the DA-GRS model. It is important to notethat the DA-GRS model does not in itself model services or applications. It rather models the core mechanisms to handle topology changes and interactions between devices, and on which applications can rest.

In this document we have provided a canonical method that covers the whole development process of such a system, from the modeling of a strategy to maintain a topological structure, to the code that adds sense to it.

In our work on the DA-GRS model we consider both the thoretical and practical aspects. From a theoretical point of view, we plan to use the model to characterize mobility classes, *i.e.* to identify what kind of basic problems (election, naming, counting, etc.) can be solved in which class, and under which assumption on mobility. From a practical point of view we are working on the design of a methodological framework for the analysis of DA-GRS algorithms, in terms of what can be guaranted with which efficiency.

Regarding the experimentation, a dedicated simulator has been developed and is freely available on the web (at [9]). We are furthermore currently porting the DA-GRS rule engine on smartphones.

References

[1] A. Casteigts and S. Chaumette, "Dynamicity Aware Graph Relabeling Systems - a model to describe MANet algorithms," in *Proceedings of the 17th IASTED International Conference on Parallel and Distributed Computing and Systems*, 2005.

[2] I. Litovsky, Y. Metivier, and E. Sopena, "Graph relabelling systems and distributed algorithms," in *Handbook of graph grammars and computing by graph transformation*, W. S. Publishing, Ed., vol. III, Eds. H. Ehrig, H.J. Kreowski, U. Montanari and G. Rozenberg, 1999, pp. 1–56.

[3] M. Bauderon, Y. Metivier, M. Mosbah, and A. Sellami, "From local computations to asynchronous message passing systems," 2002. [Online]. Available: citeseer.ist.psu.edu/bauderon02from.html

[4] J. Chalopin and Y. Métivier, "A bridge between the asynchronous message passing model and local computations in graphs," in *Mathematical Foundations of Computer Science (MFCS 2005)*, ser. Lecture Notes in Computer Science, vol. 3618. Springer-Verlag, aug 2005, pp. 212–223. [Online]. Available: http://www.labri.fr/publications/combalgo/2005/CM05

[5] Y. Métivier, N. Saheb-Djahromi, and A. Zemmari, "Randomized rendezvous," in *Colloquium on mathematics and computer science: algorithms, trees, combinatorics and probabilities*, ser. Trends in mathematics. Birkhäuser, 2000, pp. 183–194. [Online]. Available: http://www.labri.fr/publications/combalgo/2000/MSZ00

[6] Y. Metivier, N. Saheb, and A. Zemmari, "Analysis of a randomized rendezvous algorithm," *Inf. Comput.*, vol. 184, no. 1, pp. 109–128, 2003.

[7] A. Casteigts and S. Chaumette, "DA-GRS and the constraint based synchronization: a unifying approach to deal with dynamic networks," submitted to WASA'06.

[8] D. Delcampo, "Validation du modele DA-GRS sur la plateforme GRID5000, et sur terminaux mobiles communiquants," http://www.labri.fr/perso/casteigt/docs/dagrs-grid5000.ps.

[9] A. Casteigts, "SimuDAGRS, a Dynamic Network simulator for algorithms modeled by DA-GRS," available at http://www.labri.fr/perso/casteigt/simulator.html.