State-Based Behavior Specification for GCM Systems[*]

Alessandro Basso, Alexander Bolotov, Vladimir Getov

[⋆]School of Computer Science, University of Westminster, Watford Road, Harrow HA1 3TP

a.basso@wmin.ac.uk,bolotoa@wmin.ac.uk,v.s.getov@wmin.ac.uk

Abstract

This paper is in the area of automata-based formalisms of stateful systems. In particular, we have analyzed aspects of Grid systems which can be considered when developing a prototype for dynamic reconfiguration. We describe which parts of a Grid system can be utilized and translated into state-based formal specification, and how the subsequent deductive verification tests for the dynamic reconfiguration can be performed.

Introduction. The Grid Component model (GCM) developed by the CoreGRID project (see Baude et al., 2009) is a purpose-built component model to construct Grid applications. It has been adopted by the Grid Integrated Development Environment (GIDE), allowing for easy composition, monitoring and steering of Grid systems (see Basukoski et al., 2008). The features exposed in the GIDE allow us to use formal specification language and deductive reasoning verification methods in the framework of automated dynamic reconfiguration. It is essential that the properties of a Grid system, starting from the components stateful properties, and including the ones of resources (which we have collected under the umbrella of the Environment) can be monitored and reported in order to be able to comprehend the current states scenario. When monitoring a program we considered the impact on complexity that this type of specification will create in respect to execution time as well as memory consumption. We have devised a process of repetitive static analysis to minimize the impact by optimizing the program instrumentation. We can therefore monitor at runtime only a small section of the components's specification – the behavior of the stateful system – and leave the proofs of the inner functionality of primitive components' behavior to other methods, as in Barros et al. (2005). In this paper we describe how each section of the Grid system – and the environment it lays on – can be translated into formal specification and fed into a resolution based proof engine. Our final aim is to give a response to the tool we are developing within the GIDE indicating to whether the reconfiguration can take place and how.

Automata-based formalism. We began our development on the techniques expressed in Basso et al. (2008a), but changed our path to simplify development by redefining our concept of a two layers automata to a single one. We considered a simple finite state automaton on finite strings, and applied a set of specification "patterns" (following the sections described in the next section). The automata is used for the creation of labels defining various states in which the considered components and resources can be, the derived model is then directly specified in the normal form for CTL, $\mathrm{SNF_{CTL}}$, developed by Bolotov and Fisher (1999). It was shown in Bolotov et al. (2002) that a Buchi word automaton can be represented in terms of $\mathrm{SNF_{PLTL}}$, a normal form for PLTL. In a similar fashion we have represented a Buchi tree automaton in terms of $\mathrm{SNF_{CTL}}$; we use $\mathrm{SNF_{CTL}}$to specify the tree automaton and (in simulations) extend this specification to the deontic temporal specification in Basso et al. (2008b). Further, we apply a resolution-based verification technique as a verification procedure using the tool in Zhang et al. (2008).

Formalizing components and resources. When considering what parts in the GCM can be used for formal specification, we have considered four main sections, each of which follows specific criteria and can be easily fed into to a table of specification "patterns". We examine the main details below. Please not that not full specification is included for space reasons. As an example, we consider an Application (the outmost component which must be activated first) which contains 4 components Comp1 (a composite component with a sub component SubComp1.1) which is the first to be started after the application is as it is the first and only component, two components CompA and CompB running in parallel from a broadcast of Comp1 (and SubComp1.1 to start in parallel with CompA or CompB), and Comp2 a component from the gathercast of CompA and CompB.

Hierarchical Components Structure. Components in the GCM have a strict hierarchical nature. The application can then be described as: $\mathtt{start} \Rightarrow Application$ and components of the application in the form of: $Application \Rightarrow \mathbf{A}\bigcirc Comp1$, $Application \Rightarrow \mathbf{A}\Diamond Comp2$, $Comp1 \Rightarrow \mathbf{A}\bigcirc(SubComp1.1 \wedge (CompA \vee CompB))$

Inferring parallel processes from interfaces. When we consider interfaces in the GCM, we can group them in two different types: one to one, and broadcast/gathercast. In the former we have a simple connection of one server interface to a client one, while in the latter we have a single server interface which can be bound to multiple client ones.

[*]This research has been carried out under the European research and development project GridCOMP (Contract IST-2005-034442) funded partially by the European Commission.

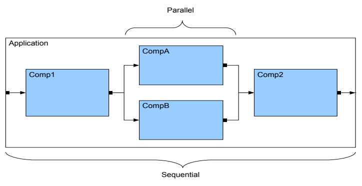

Figure 1: Sequential and Parallel Processes

In either case, interfaces can be very useful to determine whether the communication between components is carried out in a sequential or parallel matter. Imagine a component with a broadcast server interface (or several one to one server interfaces): we can easily assume that the components at the client side of those interfaces can be run in a parallel matter. On the other hand, a component which has only one server interface, can only run in a sequential matter with the component on the client interface side (see Figure: 1). Sequential specification looks like: $Comp1 \Rightarrow \mathbf{A}\Diamond Comp2$ while Parallel specification looks like: $Comp1 \Rightarrow \mathbf{A}\bigcirc(CompA \wedge CompB)$.
When in a sequential process is easy to understand that component will be started sequentially, in a parallel process, there is no real certainty - component might be all started at the next step, or first one and then the others, or perhaps none.

State of resources. When considering resources, we are able to formally specify the environment thanks to information provided in the GCM deployment file as well as other metadata information gathered at development time through a development interface. Furthermore the current state of each resource can be monitored at runtime giving us a complete picture of the resources at every given moment in time and any components that might be deployed on or requesting the use of the resource. External resources are defined as: $Comp1 \Rightarrow \mathbf{A}\Diamond Res1$. Deployment resources are defines as: $Node1 \Rightarrow Res1$ and at runtime we can have definitions like: $Res1 \Rightarrow \mathbf{A}\,\square(Comp1 \wedge Com2)$.

State of components. While the states of components could have a wide spectrum of definition points (such as *initialized, started, suspended, terminated, . . .* for the moment we can only consider the ones defined in the GCM - i.e. *started* and *stopped*. In a way this simplifies further the formalism by representing the specification as: $Comp1$ for a started component, and: $\neg Comp2$ for a stopped one.

Conclusions. The formal specification procedure introduced in this paper will be used in reconfiguration scenarios to prevent inconsistency while suggesting possible corrections to the system. While we have applied this framework to a GCM system, such procedure could be applied to other systems, giving the deductive reasoning a chance to assist other verification methods such as model checking by filling the gaps in those areas where these other well established methods cannot be used.

References

T. Barros, L. Henrio, and E. Madelaine. Verification of distributed hierarchical components. In *In Proc. of the International Workshop on Formal Aspects of Component Software (FACS'05)*, volume Electronic Notes in Theor. Computer Sci. 160, pages 41–55, 2005.

A. Basso, A. Bolotov, and V. Getov. Automata-based formal specification of stateful systems. In *In Proc. of Automated Reasoning Workshop*, 2008a.

A. Basso, A. Bolotov, and V. Getov. *Behavioural Model of Component-based Grid Environments.*, volume From Grids To Service and Pervasive Computing, pages 19–30. Springer, 2008b.

A. Basukoski, V. Getov, J. Thiyagalingam, and S. Isaiadis. *Component-based Development Environment for Grid Systems: Design and Implementation.*, volume In: Making Grids Work, pages 119–128. Springer, 2008.

F. Baude, D. Caromel, C. Dalmasso, M. Danelutto, V. Getov, L. Henrio, and C. Pérez. *GCM: A Grid Extension to Fractal for Autonomous Distributed Components.*, volume Annals of Telecommunications, vol. 64(1-2), pages 5–24. Springer, 2009.

A. Bolotov and M. Fisher. A clausal resolution method for ctl branching time temporal logic. *Journal of Experimental and Theoretical Artificial Intelligence.*, 11:77–93, 1999.

A. Bolotov, C. Dixon, and M. Fisher. On the relationship between normal form and w-automata. *Journal of Logic and Computation, Oxford University Press.*, Volume 12(Issue 4):561–581, August 2002.

L. Zhang, U. Hustadt, and C. Dixon. First-order resolution for ctl. Technical Report ULCS-08-010, Department of Computer Science, University of Liverpool, 2008.