Specification and Verification of Reconfiguration Protocols in Grid Component Systems

Bolotov, A., Getov, Vladimir, Basso, A. and Basukoski, A.

A paper published in the proceedings of the Workshop on Automated Reasoning: Bridging the Gap between Theory and Practice, London, 30 - 31 Mar 2010, University of Bristol Technical Report.

Specification and Verification of Reconfiguration Protocols in Grid Component Systems

Alessandro Basso*, Alexander Bolotov, Artie Basukoski, Vladimir Getov
*Harrow School of Computer Science
University of Westminster
United Kingdom
A.Basso@student.westminster.ac.uk

Ludovic Henrio†
†CNRS/I3S/INRIA
Sophia Antipolis
France
ludovic.henrio@sophia.inria.fr

Mariusz Urbanski‡
‡Institute of Psychology
Adam Mickiewicz University
Poznan
Poland
murbanski@amu.edu.pl

Introduction

There are two approaches to building long-lived and flexible Grid systems: exhaustive and generic. The former approach provides rich systems satisfying every service request from applications but consequently its implementation suffers from very high complexity. In the latter approach, we represent only the basic set of services (minimal and essential) and thus overcome the complexity of the exhaustive approach. However, to achieve the full functionality of the system, we must make this lightweight core platform reconfigurable and expandable. One of the possible solutions here is to identify and describe the basic set of features of the component model and to consider any other functions as pluggable components which can be brought on-line whenever necessary (9).

Establishing the theoretical foundations of the generic processes involved in designing and functioning of such Grid systems is highly important. Significant part of this research lies in the area of formal specification and verification of the core component model and the properties of the desired Grid systems.

Fractal Component Model

We depart from the Fractal component model (6), a modular and extensible component model. The Fractal specification defines a set of notions necessary for the model, an API (Application Program Interface), and an ADL (Architecture Description Language).

Components are characterized by their *content* and the *membrane*. If the content of a component is hidden (in which case it is simply a black box) we call it *primitive*. If a component is constituted by a system of some other components (sub-components) it represents a *composite* component controlled by the membrane.

Fractal is a multi-level specifications. Depending on their conformance level, Fractal components can feature introspection and/or configuration. The *control* interfaces are used in the Fractal model to allow configuration (reconfiguration), and are defined as *non functional*. A *functional* interface can provide the required functionalities and we call it the *server* interface. Alternatively, a *client* interface requires some other functionalities. Component interfaces are linked together by *bindings*.

Reconfiguration is obtained by triggering appropriate actions on three interfaces: life-cycle, binding, and content control interface. A reconfiguration can be triggered by any component that has a reference to a correct non-functional interface.

In this work we focus on predefined categories of reconfigurations and on proving properties on these reconfiguration. As far as the reconfiguration is concerned we use the classical assumption that replacing a component by a similar one is safe for the system.

Specification of the Fractal component model in temporal logic

We provide the formal framework for the components in a specific branching-time temporal logic, or SNF $_{\mathrm{CTL}}$(Separated Normal Form for Computation Tree Logic) (5), and then directly apply temporal resolution as a deductive verification tool.

Taking into account potential applications of CTL-type logics in specification and verification of concurrent and distributed systems, two combinations of future time temporal operators, $\Diamond$ ('sometime') and $\Box$ ('always'), are useful in expressing *fairness* (7): $\Diamond\Box p$ (p is true along the path of the computation except possibly some finite initial interval of

it) and $\Box\Diamond p$ (p is true along the computation path at infinitely many moments of time). It has been shown that SNF $_{\mathrm{CTL}}$ can express these simple fairness constraints and their Boolean combinations (2; 3). In (1; 2) a clausal resolution over the set of clauses SNF$_{\mathrm{CTL}}$ has been defined while the search strategies for this method were presented in (4).

These developments allow us to argue on behalf of the following formal framework to reason about the configurations/reconfigurations protocols of a Grid component model:

$$FCM \longrightarrow ECTL^{+} \longrightarrow SNF_{ctl}(FCM) \longrightarrow BTR$$

Here we suggest the specification of the Fractal component model (FCM) in the logic $ECTL^{+}$ with the subsequent translation into SNF$_{\mathrm{CTL}}$(FCM), the SNF$_{\mathrm{CTL}}$ based formal specification of FCM, and application of the 'branching temporal resolution' method (BTR), the temporal resolution technique defined over the set of SNF $_{\mathrm{CTL}}$ clauses.

Example Specification

Let us consider a simple printing queue component model which consists of a client and one printing queue component as primitives. The client interfaces of the client are of type CI_a and the server interfaces of the printing queue are of type SI_r. In the fractal model four controller interfaces are defined, for reasons of space, we will only specify the safe-unbinding part of a reduced Life-Cycle Controller (LCC) so that it can be used in the deductive reasoning (Figure 1).

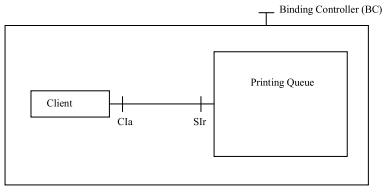

Figure 1: Example in Fractal

Dealing with non-functional interface. Let the propositions $Prim_1, \ldots, Prim_n$ refer to all our primitive controllers. Each $Prim_i$ has a form $Prim_n(CI_a, SI_r)$, i.e. when true it specifies that a Primitive with Client Interface CI_a is bound to the Server Interface SI_r. In our example we have two primitives, one for the Printing Queue and one for the Client using the Printing Queue.

LCC is a proposition which when true signifies that the Life Cycle Controller is active.

Before introducing the Life Cycle Controller Formula we would need to specify how components are started and stopped. Here we will only provide a partial specification of the Life Cycle Controller and two primitive components; we only deal with the formula that captures the bindings of the two components. We will model the start of the components by attaching them to **start**.

Now we introduce the specification of the Simplified Life-Cycle Controller:

$$\neg LCC \wedge \neg(Prim_1(CI_a, SI_r) \vee Prim_2(CI_a, SI_r)) \Rightarrow$$
$$\mathbf{A}\Box LCC \Rightarrow (Prim_1(CI_a, SI_r) \wedge Prim_2(CI_a, SI_r))$$

which states that if neither of the components are bound and the LCC is not active then in all possible computations when the LCC is active then we must have the two components bound.

Specification of reconfiguration scenarios.

When a controller needs to dynamically reconfigure the scenario, some properties that we assumed as always true may not hold anymore, for example a component might be removed from the model, or another added, making the system to have to take into consideration other possible paths which have just opened.

The Client can set a request for printing of the type $req(CI_a, SI_r)$. When true, this proposition states that a printing request has been raised by the client which has binding (CI_a, SI_r). The Printing queue specification is represented as $print(CI_a, SI_r)$ which states that a printing request has been satisfied by the printer which has binding (CI_a, SI_r). We will take in consideration the safety part of the specification and its requirements (8). Using the specifications of the components we are able to generate a complete specification for the example considered. To verify the properties of this specification we extract relevant sets of SNF$_{\mathrm{CTL}}$ clauses (3) and then apply a clausal temporal resolution method (5).

References

[1] A. Bolotov. Clausal resolution for extended computation tree logic ECTL. In *Proceedings of the Time-2003/International Conference on Temporal Logic 2003*, Cairns, July 2003. IEEE.

[2] A. Bolotov and A. Basukoski. Clausal resolution for extended computation tree logic ECTL. *Journal of Applied Logic*, in Press.

[3] A. Bolotov and A. Basukoski. A Clausal Resolution Method for Branching Time Logic ECTL+. To be published in Annals of Mathematics and Artificial Intelligence, Springer Verlag.

[4] A. Bolotov and A. Basukoski. Search Strategies for Resolution in CTL-Type Logics: Extension and Complexity. In *Proceedings of the 12th International Symposium on Temporal Representation and Reasoning, (TIME 2005)* 195 - 197, IEEE Computer Society, 2005.

[5] 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.

[6] E. Bruneton, T. Coupaye, and J. Stefani. Recursive and Dynamic Software Composition with Sharing. In *Proc. of the 7th Int. Workshop on Component-Oriented Programming (WCOP02)*, 2002.

[7] E. A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science: Volume B, Formal Models and Semantics.*, pages 996–1072. Elsevier, 1990.

[8] Z. Manna and A. Pnueli. Temporal Specification and Verification of Reactive Modules. Weizmann Institute of Science Technical Report, March 1992.

[9] J. Thiyagalingam, S. Isaiadis and V. Getov. Towards Building a Generic Services Platform: A Components- Oriented Approach. In: V. Getov and T. Kielmann, eds. Component Models and Systems for Grid Applications, Springer-Verlag, 2004.