Behavioral Analysis of the Agent-Based Community Grid Solution for the Large Hadron Collider beauty Experiment

Daniela Remenska[1,3], Tim A.C. Willemse[2],
Henri Bal[1], Kees Verstoep[1], Wan Fokkink[1],
Jeff Templon[3]
[1]Dept. of Computer Science, VU University Amsterdam, The Netherlands
[2]Dept. of Computer Science, TU Eindhoven, The Netherlands
[3]NIKHEF, Amsterdam, The Netherlands

***Abstract*—DIRAC (Distributed Infrastructure with Remote Agent Control) is the Grid solution designed to support production activities as well as user data analysis for the Large Hadron Collider beauty experiment. It consists of cooperating distributed services and a plethora of light-weight agents delivering the workload to the Grid resources. Services accept requests from agents and running jobs, while agents actively fulfill specific goals. Services maintain database back-ends to store dynamic state information of entities such as jobs, queues, or staging requests. Agents continuously check for changes in the service states, and react to these accordingly. The logic of each agent is rather simple; the main source of complexity lies in their cooperation. These agents run concurrently, and communicate using the services' databases as a shared memory for synchronizing the state transitions. Despite the effort invested in making DIRAC reliable, entities occasionally get into inconsistent states. Tracing and fixing such behaviors is difficult, given the inherent parallelism among the distributed components, and the size of the implementation.**

In this paper we present an analysis of DIRAC with mCRL2, process algebra with data. We have reverse engineered two critical and related DIRAC subsystems, and subsequently modeled their behavior with the mCRL2 toolset. This enabled us to easily locate race conditions and livelocks which were confirmed to occur in the real system. We further formalized and verified several behavioral properties of the two modeled subsystems.

Keywords-DIRAC, service-oriented architecture, agents, stager, mCRL2, model checking, process algebra

I. INTRODUCTION

The Large Hadron Collider beauty (LHCb) experiment [1] is one of the four large experiments conducted on the Large Hadron Collider (LHC) accelerator, built by the European Organization for Nuclear Research (CERN). Immense amounts of data are produced at the LHC accelerator, and subsequently processed by physics groups and individuals worldwide. The sheer size of the experiment is the motivation behind the adoption of the Grid computing paradigm. The Grid storage and computing resources for the LHCb experiment are distributed across several institutes in Europe. To cope with the complexity of processing the vast amount of data, a complete Grid solution, called DIRAC (Distributed Infrastructure with Remote Agent Control) [2], [3], has been designed and developed for the LHCb community.

DIRAC forms a layer between the LHCb community of users and the heterogeneous Grid resources, in order to allow for optimal and reliable usage of these resources. It consists of many cooperating distributed services and agents which deliver workload to the resources. In particular, services are passive modules which accept incoming requests from agents and running jobs. Each service has an associated database back-end to store dynamic state information of entities such as jobs, queues, or staging requests. Agents are light-weight independent components that fulfill specific system functions. Using a polling strategy they continuously observe for updates in the service states and react accordingly. The logic of each individual agent is relatively simple; the overall system complexity emerges from the cooperation among them. Namely, these agents run concurrently, and communicate using the services' databases as a shared memory (blackboard paradigm [4]) for synchronizing the entities' state transitions.

Although much effort is invested in making DIRAC reliable, entities occasionally get into inconsistent states, leading to a potential loss of efficiency in both resource usage and manpower. Debugging and fixing the root of such encountered behaviors becomes a formidable mission due to multiple factors: the inherent parallelism present among the system components which are deployed on different physical machines, the size of the implementation (around 150000 lines of Python code), and the distributed knowledge of different subsystems within the collaboration.

In this paper we propose the use of more rigorous (formal) methods for improving software quality. Model checking [5] is one such technique for analysis of an abstract model of a system, and verification of certain system properties of interest. Unlike conventional testing, it allows full control over the execution of parallel processes and also supports fully automated exhaustive state-space exploration.

We used the mCRL2 language [6] and toolset [7] to model the behavior of two critical and related DIRAC components: the workload management (WMS) and the storage management system (SMS). Based on Algebra of Communicating Processes (ACP) [8], mCRL2 is able to deal with generic data types as well as user-defined functions for data transformation. This makes it particularly suitable for modeling the data manipulations made by DIRAC's agents. Visualizing the state space and replaying scenarios with the toolkit's simulator enabled us to gain insight into the system behavior, incrementally improve the model, and to already detect critical race-conditions and livelocks, which were confirmed to occur in the real system. Some of them were a result of simple coding bugs; others unveiled more elementary design problems. We further formulated, formalized and verified several general and application-specific properties.

The idea of modeling existing systems using formal techniques is as such not new. Earlier studies ([9], [10], [11], [12], [13], [14], [15]) mostly focused on modeling and verifying hardware or communication protocols, since the formal languages and tools at hand were not sufficiently mature to cope with more complex data-intensive distributed systems. More recently, success stories on modeling real-life concurrent systems with data have been reported **(**[16]**,** [17]**,** [18]**,** [19]**,** [20]**).** In [18] the authors have implemented a tool for automatic translation of the SystemC language into mCRL2 statements. This greatly simplifies the analysis, but has so far been feasible only when the language of implementation is domain-specific, or alternatively, a reasonably small subset of a general-purpose language is considered for translation. The only exception in this respect is the Java Pathfinder tool [19] used to find deadlocks and other behavioral properties in Java software systems developed by NASA.

We believe that the challenges and results of this work are unique in a number of aspects. First, to the best of our knowledge, the code-base and the number of concurrent components engaged in providing DIRAC's functionality considerably outnumber previous industrial cases. Second, the choice of Python as implementation platform has lead to prevailing usage of dynamic structures (whose types and sizes are determined at runtime) throughout DIRAC, challenging the transition to an abstract formal representation. We have nevertheless established general guidelines on extracting a model outline from the implementation. Third, analysis of this kind is typically performed after a problem has already surfaced in the real system, as a means to understand the events which lead to it and test for possible solutions. We managed to stumble on an actual bug at the same time it was observed in practice, which increased our confidence in the soundness of the model.

The paper is organized as follows. Section 2 introduces the architecture of DIRAC, focusing on the two studied subsystems. Section 3 gives a brief overview of the mCRL2 language, and describes our approach to abstracting and modeling the behavior of these subsystems. Section 4 presents the analysis with the mCRL2 toolset and the issues detected. Section 5 concludes and discusses future work.

II. DIRAC:A Community Grid Solution

A. *Architecture Overview*

The development of DIRAC started in 2002 as a system for production of simulation (Monte Carlo) data that would serve to verify theory, aspects of the LHCb detector design, as well

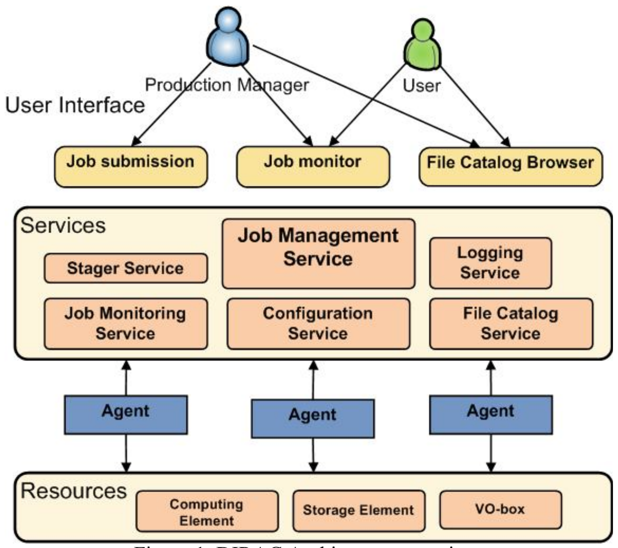

Figure 1. DIRAC Architecture overview

as to optimize algorithms. It gradually evolved into a complete community Grid solution for data and job management, based on a general-purpose framework that can be reused by other communities besides LHCb. Over the years it has been considerably reengineered in order to meet the requirements for processing the real data coming from the LHC. Today, it covers all major LHCb tasks starting with the raw data transfer from the experiment's detector to the Grid storage, several steps of data processing, up to the final user analysis.

DIRAC follows the Service Oriented Architecture (SOA) paradigm, accompanied by a network of lightweight distributed agents which animate the system. Its main components are depicted in Figure 1.

The *services* are passive components which react to requests from their clients, possibly soliciting other services in order to fulfill the requests. They run as permanent processes deployed on a number of high-availability hosts (VO-boxes) at CERN, and persist the dynamic system state information in database repositories. To provide a certain level of load balancing and redundancy, multiple mirror instances of each service can be deployed on different machines. The user interfaces, agents or running jobs can act as clients placing the requests to DIRAC's services. All clients and services are built in the DISET framework [21] which provides secure access and flexible authorization rules.

Agents are active components that fulfill a limited number of specific system functions. They are built around a common framework which provides a uniform way for configuration, deployment, control, and logging of each agent activity. They can run in different environments, depending on the nature of their mission. Some are deployed close to the corresponding services, while others run on the Grid worker nodes. Examples of the later are the so-called Pilot Agents which are part of the Workload Management System, explained in more details in the following section B. All DIRAC agents repeat the same logic in each iteration cycle: they observe for changes in the service states, and react accordingly by initiating actions (like job submission or data transfer) which may update the states of various system entities.

Resources are software abstractions of the underlying heterogeneous Grid computing and storage resources allocated to LHCb, providing a uniform interface for access to them.

The DIRAC functionality is exposed to users and developers through a rich set of command-line tools forming the DIRAC API, complemented by a Web portal for visual monitoring the system behavior and controlling the ongoing tasks. Both the Web and command-line *interfaces* ensure secure system access using X509 certificates.

B. *Workload Management System*

tbw

C. *Storage Management System*

tbw

References

[1] Large Hadron Collider beauty experiment. [Online]. http://lhcb-public.web.cern.ch/lhcb-public/

[2] A. Tsaregorodtsev et al., "DIRAC: A Community Grid Solution," in *CHEP*, vol. 119, 2007.

[3] A. C. Smith and A. Tsaregorodtsev, "DIRAC: Reliable Data Management for LHCb," in *CHEP*, 2007.

[4] J.W. McManus and W.L. Bynum, "Design and analysis techniques for concurrent blackboard systems," *Systems, Man and Cybernetics*, vol. 26, no. 6, pp. 669 - 680, Nov 1996.

[5] J. F. Groote and T. A.C. Willemse, "Model-checking processes with data," *Science of Computer Programming*, vol. 56, no. 3, May/June 2005.

[6] J. F. Groote, A. Mathijssen, , M. Reniers, Y. Usenko, and M Van Weerdenburg, The formal specification language mCRL2.

[7] J. F. Groote et al., "The mCRL2 toolset," Dept. of Mathematics and Computer Science, Eindhoven University of Technology, 2008.

[8] J.C.M Baeten, T. Basten, and M.A. Reniers, *Process Algebra: Equational Theories of Communicating Processes*, 1st ed. New York, NY, USA: Cambridge University Press, 2009.

[9] Y.T. He and R. Janicki, "Verifying protocols by model checking: a case study of the wireless application protocol and the model checker SPIN," in *CASCON '04 Proceedings*.

[10] K. Palmskog, "Verification of the session management protocol," School of Computer Science and Communication, Royal Institute of Technology, Stockholm, Master's Thesis 2006.

[11] G.J. Holzmann, *Design and validation of computer protocols*. Upper Saddle River, NJ, USA: Prentice-Hall Inc., 1990.

[12] W. Fokkink, *Modelling Distributed Systems: Protocol Verification with µCRL*, 2nd ed.: Springer, 2011.

[13] T. Ball and S.K. Rajamani, "The SLAM Toolkit," in *Computer Aided Verification (CAV) Proceedings*, 2001, pp. 260-264.

[14] B. Badban, W. Fokkink, J.F. Groote, J. Pang, and J. Van de Pol, "Verification of a sliding window protocol in µCRL and PVS," *Formal Aspects of Computing*, vol. 17, no. 3, October 2005.

[15] S.M.S. Islam, M.H. Sqalli, and S. Khan, "Modeling and Formal Verification of DHCP Using SPIN," *IJCSA*, pp. 145-159, May 2006.

[16] J.L. Hwon, V.J.J. Kuster, and T.A.C. Willemse, "Analysing the Control Software of the Compact Muon Solenoid Experiment at the Large Hadron Collider," in *FSEN2011*, in press.

[17] D. Bošnački, A. Mathijssen, and Y.S. Usenko, "Behavioural Analysis of an I2C Linux Driver," *14th International Workshop on Formal Methods for Industrial Critical Systems*, 2009.

[18] H. Hojjat, M.R. Mousavi, and M. Sirjani, "Formal Analysis of SystemC Designs in Process Algebra," *Fundamenta Informaticae*, pp. 19-42, 2011.

[19] V. Visser and P.C. Mehlitz, "Model Checking Programs with Java PathFinder," in *SPIN*, 2005.

[20] B. Ploeger, "Analysis of ACS using mCRL2," Technische Universiteit Eindhoven, CS-Report 09-11 2009.

[21] Bargiotti M,Brook N,Casajus Ramo A,Castellani G,Charpentier Ph,Cioffi C,Closier J,Graciani Diaz R,Kuznetsov G,Y Li Y,Nandakumar R,Paterson S,Santinelli R,Smith A C,Seco Miguelez M,Gomez Jimenez S Tsaregorodtsev A, "DIRAC: A Community Grid Solution," in *CHEP*, vol. 119, 2007.

[22] Andrew C. Smith and Andrei Tsaregorodtsev, "DIRAC: Reliable Data Management for LHCb," in *CHEP*, 2007.

[23] M. Van Eekelen, S. Ten Hoedt, R. Schreurs, and Y.S. Usenko, "Analysis of a session-layer protocol in MCRL2: verification of a real-life industrial implementation," in *FMICS'07 Proceedings of the 12th international conference on Formal methods for industrial critical systems*, Berlin, Heidelberg, 2007.

[24] J. F. Groote, A.H.J. Mathijssen, M.A. Reniers, Y. S. Usenko, and M.J. Van Weerdenburg, "Analysis of distributed systems with mCRL2," in *Process Algebra for Parallel and Distributed Processing*.: Chapman and Hall/CRC, 2009, ch. 4.

[25] S. Chandra, P. Godefroid, and C. Palm, "Software model checking in practice: an industrial case study," in *Proceedings of the 24th ICSE*, 2002, pp. 431-441.

[26] G.J. Holzmann and M.H. Smith, "Automating software feature verification," *Bell Labs Technical Journal*, 2000.

[27] S. Blom, J. Van de Pol, and M. Weber, "LTSmin: Distributed and symbolic reachability," in *Computer Aided Verification (CAV) Proceedings*, 2010, pp. 354–359.