

Domain-Specific Metadata for Model Validation and Performance Optimisation

Jeyarajan Thiyagalingam¹, Vladimir Getov¹,

Sofia Panagiotidi², Olav Beckmann², John Darlington²

*¹Harrow School of Computer Sciences, University of Westminster,
Watford Road, Northwick Park, Harrow HA1 3TP, United Kingdom*

²Department of Computing, Imperial College London, London SW7 2AZ, U.K.



CoreGRID Technical Report
Number TR-0068
January 23, 2007

Institute on Grid Systems, Tools and Environments

CoreGRID - Network of Excellence
URL: <http://www.coregrid.net>

Domain-Specific Metadata for Model Validation and Performance Optimisation

Jeyarajan Thiyagalingam¹, Vladimir Getov¹,
Sofia Panagiotidi², Olav Beckmann², John Darlington²

¹Harrow School of Computer Sciences, University of Westminster,
Watford Road, Northwick Park, Harrow HA1 3TP, United Kingdom

²Department of Computing, Imperial College London, London SW7 2AZ, U.K.

CoreGRID TR-0068

January 23, 2007

Abstract

Interactive problem solving environments are gaining widespread acceptance within the Grid community. While developing applications and frameworks to support and to integrate with these interactive environments, it is also necessary to treat the legacy applications with the same importance. In this paper, we report an ongoing effort in enabling a legacy application to support, integrate with and to evolve as a scalable problem solving environment.

Our strategy is to componentise the existing legacy framework, augment each component with metadata, and using the metadata to address different issues arising when performing component composition. We emphasise on the issues relating to the metadata and describe the means for specifying, publishing and using the metadata in solving two different but crucially important issues when performing component composition, model validation and model efficiency.

1 Introduction

Interactive problem solving environments are gaining increasing interest from end users and finding their widespread acceptance within the Grid community. The intuitive part of such frameworks is how the complexities of solving computationally demanding problems are hidden from the end-users by visually appealing front-ends.

Component-based programming is a suitable methodology for developing applications and frameworks to support, to integrate with and to evolve as a problem solving environment. In the context of component-oriented programming, a framework or an application is seen as a composition of components, on which some of them might have been developed outside the context of the application domain. As a result, when composing an application from components, performing a strict validation on the bindings of interfaces and semantics of the composition are important to guarantee that the composition is functionality valid. Although this process has been relatively simplified with the modern programming languages, when considering legacy applications or frameworks, a number of issues still remain to be addressed.

Majority of legacy applications do not have any explicit notion of components and performing a composition based on these legacy codes is often complicated. Although functional aspects of legacy-code can be componentised without any explicit modernisation (such as rewriting in an object-oriented language), such componentised versions do not match their counterparts in many aspects.

We argue that when each component is augmented with additional information, metadata, the task of performing valid and efficient compositions becomes relatively simplified and can be automated. Such an approach involves appropriately specifying, publishing, extracting and correctly using the metadata for different operations. The (specification

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

of the) metadata for a component highlights all salient features of a component. This information can voluntarily be embedded by developers in the form of annotations [6, 10, 5, 12] or by the compilers. Though compilers may capture and provide substantial amount of information, high-level, domain-specific details are better captured by manual specification or by specialised tools. Metadata for a component can be furnished as part of the binary or externally. For instance, for a selected class of binaries, such as Java byte-code or .NET-based binaries [11], these metadata can be examined or extracted using reflective introspection [18, 17]. Where such facilities are infeasible or limited, for instance when the metadata cannot be embedded, the metadata has to be furnished separately. Once the metadata for a component is extracted (either through reflective introspection mechanisms or by examination of external metadata), the information can be staged to address different issues.

However, legacy software pose many challenges in all these aspects. Lack of support for appropriately expressing components, limited techniques for specifying and extracting metadata, and limited availability of well defined mechanisms for using the metadata contribute towards this problem.

In this paper, we report our findings in addressing these issues, using a legacy application as a driving example. Although we use the legacy-code based case study as a motivating example to illustrate our techniques, the techniques are equally applicable to modern component-oriented solutions and across many different application domains. The key step is to find methods for specifying, publishing and extracting the metadata from legacy codes. Techniques for using the metadata to address different issues are the same across legacy and contemporary systems. We illustrate how we plan to use component-specific metadata to address two interesting problems arise when performing component composition, especially when evolving a legacy code-base into an interactive problem solving environment. The main contributions of this paper are as follows:

1. We formulate methods for specifying, publishing and extracting metadata for/from legacy software components
2. We present a number of complementary strategies for verifying the validity of compositional patterns
3. We discuss our plan in using the metadata for performing efficient component composition by using a real-world component-based application.

The rest of this paper is organised as follows: In Section 2 we describe our underlying example, GENIE - a component based modular platform for simulating long term evolution of Earth's climate. Section 3 describes the motivation for our work. The overall mechanism for specifying, publishing and extracting metadata are discussed in Section 4. Section 5 discusses how the metadata could be used to address two different issues: verification of validity of composition and performing efficient component compositions. In Section 6, we discuss previous work immediately related to ours and Section 7 concludes the paper with directions for further research.

2 GENIE Application

2.1 Overview

GENIE (Grid ENabled Integrated Earth system modelling) [1] is a scalable modular platform used to produce and simulate a range of Earth system models in order to understand the long-term changes in Earth's climate. The framework includes different variants of *earth modules* such as the atmosphere, ocean, sea-ice, marine sediments, land surface, vegetation and soil and ice sheets. A typical Earth system model is a composition of different variants of these earth modules (or different instances of earth modules), which are in fact different *configurations* suited for different simulation scenarios. Each Earth module is bound to the natural laws of physics that govern the exchange of energy, fluxes, water and other physical or biogeochemical tracers. Further, each Earth module uses their own module-specific computational models and module-specific data representations for representing boundary surfaces where the exchanges occur.

Figure 1 shows a sample Earth system model (or configuration). The model uses different instances of each of the Earth system modules (instance types are shown within parenthesis). Interactions between different (instances of) Earth modules are shown by arrows. In addition to Earth system modules, a configuration may also use modules to represent exchanged or accumulated physical or biogeochemical tracers (for example, Surface flux in the Figure).

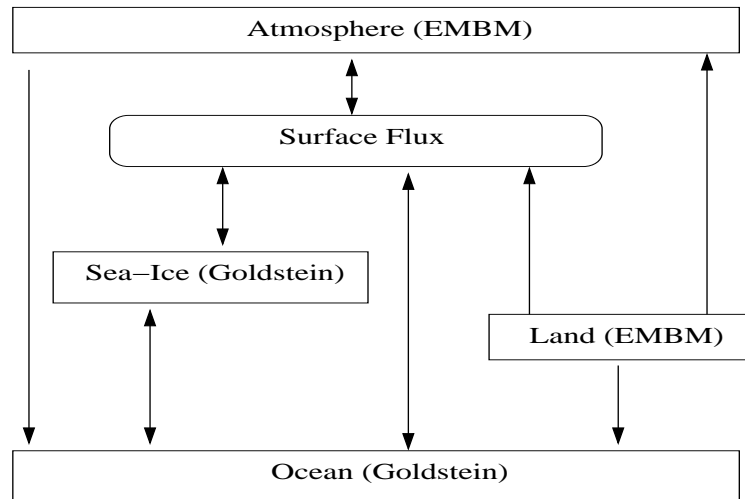


Figure 1: A sample configuration representing an Earth system model

2.2 The Past, the Present and the Future

In the original implementation, the GENIE was implemented in pure FORTRAN without any explicit notion of components and services. The framework included a driver module for orchestrating the execution, inter- and intra-module communications and data exchanges for a given configuration.

At present, these earth-modules have been componentised. To facilitate inter-operability between different language-specific implementations and to expose these components on the Web-Services front, these components have been wrapped through Java interfaces [13]. With these wrappers in place, the overall composition of these components (and thus different earth system models) may include remote components and remote services.

Currently, the simulation of a given model could be submitted, managed and executed on the Grid using the Globus Toolkit [7] or Condor [16].

In the context of Grid, problem solving environments or problem solving portals are finding broader range of audience including scientists. Evolving GENIE as a, or as a part of a portal is an interesting challenge. This includes creating component repositories, building automated submission, management, scheduling and execution mechanisms and building/supporting visualisation platforms. In addressing this challenge, although we could potentially make use of existing techniques to handle different issues which may arise within, there are still issues need to be addressed. We use one of these issues, discussed in Section 3 as a key motivation for this paper and to illustrate our techniques.

3 Motivation

One of our goals in improving the GENIE platform to evolve the same as a or as a part of a portal where end-users could visually compose a simulation model. Such a visual composition will include different visual components representing various earth modules, which may be web services or native components. Each component is permitted to have different variants (with varying models, different parameterised configuration, costs, locations, performance and interfaces) whose properties can be configured, at least partially. In effect, the overall properties of a composition, i.e. the configuration of the model, can be configured by end-users. There are at least two immediate issues arising in supporting such user configurable simulation models. Firstly, the validity of a composed simulation model should be verified against the law of nature, or more specifically against the domain-specific facts of Earth system modelling. Secondly, the composition should be made efficient in terms of cost, performance and usage of resources.

The task of verifying the validity of a composition requires inherent knowledge of components (such as applicability information, parameters, their ranges, patterns of interactions and other similar information) and requires substantial exposure to the domain-specific details of Earth system modelling. Although it is possible to embed these information as part of the application, such an approach would prevent the scalability issue to be handled efficiently,

i.e. components cannot be added or removed dynamically. Decoupling the platform from possessing any component-specific information (component metadata) leads us to furnish the same metadata through an external source. However, in the current setup of GENIE, components cannot carry any form of metadata in their own.

Secondly, when composing components to build a model, opportunities may arise for improving the efficiency of compositions. Efficiency of a composition is a multivariate function. However, we restricted our cost model to have only one parameter, namely runtime (performance), to simplify the process. Opportunities for optimisation may also arise during runtime and using the metadata may result in significant improvement in performance. For example, performance is highly correlated to the location of web-services and choosing a service whose proximity is closer to the client has desirable advantages on performance.

A closer inspection of these requirements show that if each component is augmented with metadata, it is at least partially possible to overcome some of the challenges. We discuss the nature and specification of the metadata in Section 4 and our means to exploit the metadata in Section 5.

4 Metadata for Legacy Components

The specification of the metadata should permit information to be equally expressed across different components while permitting salient and domain-specific features of components to be highlighted. Towards this, the metadata should capture any implementation-specific and composition-specific information and details relating to the domain of the application. The organisation of the metadata, in terms of management and usage, is also equally important to the contents of the metadata. We follow the organisational structure outlined in [17] to organise our metadata. Metadata related to components are externally supplied as discussed below (in contrast to the approach of being supplied by the binary itself).

In our case, we manually specified the metadata for each componentised version of the legacy-code. The metadata is then associated to the matching component and placed in an associated repository. Service enabling our componentised versions results in additional advantage of associated metadata being served as part of the services contract. When not accessed through services, the external metadata is supplied by explicit method calls produced by component wrappers.

The exact information to be captured/specified can be divided in three different groups: basic component-specific information, domain-specific information and experience-specific information. However, in terms of implementation, it may not be entirely possible to avoid any overlaps or groupings from occurring. The component-specific information specifies the interfaces and interactions to the component such as inputs, outputs, their types and other compositional information (such as containment relationships and parent module information). The domain-specific information are often in the form of constraints, specifying valid ranges, valid values, valid units and additional conditions which are fully derived through the knowledge of the component. Finally, experience-specific information are accumulated across runs.

5 Staging the Metadata

The next step is to appropriately use the metadata to address different problems. In our case, we are considering two different issues: model validation and efficient composition, which are discussed in the following sub-sections.

5.1 Model Validation

In an Earth simulation model, different instances relating to different Earth modules may coexist. This means, for instance, there may be two different atmospheric models may coexist inside a single simulation model, perhaps to represent different atmospheric conditions over different regions of the Earth. Each instance interacts with each other for exchanging physical and/or biogeochemical tracers, specifically through their ports. Since different instances may share common interface properties, validating interface-specific details alone may not be sufficient enough to guarantee that the model is valid. We use the different parts of metadata in the process of validating a model.

- **Physical units of ports:** Wherever applicable, each physical tracer (such as speed, energy) and/or biogeochemical tracer (such as CO₂, dust, Alkalinity) is related to a unit (such as K , ms^{-2}). When verifying a coupling

between instances of different modules, their physical and biogeochemical couplings need to be verified to ensure that the connections between tracers of coupled modules are valid.

- **Resolution/Dimension of computational models:** As outlined earlier, each component may be configured to use a specific computational model, for instance the resolution of spatial grids and/or dimension of the spatial grids. In many cases, these models do not match and this issue is appropriately handled depending on the context. For example, when spatial grids are different, tracer values are appropriately interpolated/extrapolated. However, this often comes with correction factor to guarantee the conservation of energy/fluxes. Validating whether a coupling leads to large deviation in values or corrections may result in verifying whether there are violations of physical laws of conservation.
- **Number of instances:** An Earth module may mandate an upper or lower bound on number of instances that may be present in a model. Verifying that the instances do not violate this requirement, provides an extra scoring.
- **Value range:** A possible range of values that an input/output tracer can assume is specified as part of the metadata, for example value range for temperature. Depending on the model, the value range may help arresting invalid models at pre-simulation stage or during simulation.
- **Existing models/couplings:** Capturing and recording outcomes of models and couplings as part of the metadata (as part of the experience-specific metadata) may help in using experience-specific data in identifying invalid models/couplings.
- **Interface- and type-specific information:** In addition to existing details, interface- and type-specific details of coupled tracers need to be matched.

We are continuing our effort in identifying more domain-specific metadata parameters to be used against validation of simulation models. An added advantage of using the component metadata is that it partly guarantees that the application (or component) is free from value specific logics.

5.2 Performance Optimisation

An Earth system model is simulated for a long period, for instance for multi-millennial periods, and it is both time consuming and computationally intensive. Although each module may be optimised for best performance (or for other objective functions), the overall composition cannot be pre-optimised. One of the reasons for a composition to yield sub-optimal performance is that components are not aware of the overall behaviour of the composition. The use of the metadata in performing efficient component composition has previously been demonstrated by many authors [10, 4]. The key idea in our approach is to tailor the metadata to include domain-specific and experience-specific information in addition to the component/interface-specific details. We use the following metadata for resolving key performance issues.

- **Location/type of component:** As outlined already, a composition may include locally available, native components and remote web-services. Since, the communication latencies are highly correlated to the overall performance of the composition, wherever applicable, locally available components should be preferred to web-services and services in close proximity to services with high latencies. However, this issue may be further complicated if a second optimisation parameter is added, such as cost.
- **Computational models:** Some of the computational models of a given component may have matching computational model such that they all yield almost similar results but exhibiting significant difference in performance. For example, when simulating maximum Atlantic meridional overturning circulation (MOC), resolutions of $36 \times 36 \times 8$ and $72 \times 72 \times 16$ for oceans, results in similar MOC figures for higher flux corrections. Further, the choice of a faster resolution may be justified by subsequent interpolations/extrapolations for corrections.
- **Data from past runs:** Instance of some Earth modules always have similar (or constant) startup or input values. Wherever possible, the past data can be used to save computational time. Very frequently, similar simulation models may overlap in time period and thus the data. For instance, exploring the stability of the ocean thermohaline circulation (THC) is very repetitive in nature and it is possible that the simulation span to overlap with an existing (similar) model.

We are investigating the issue of staging these metadata to improve the overall performance of simulation models, THC in particular.

6 Related Work

Problem solving and modular visualisation environments and work-flow editors for composing operations are based on providing high level abstract view of components/modules for exploring problem/solution spaces [8, 4, 2]. Our work is aligned with similar interests to theirs, but we make extensive use of component and domain-specific metadata to simplify the process of our goals.

The Bespoke Framework Generator (BFG) [14] is a prototype implementation of the Flexible Coupling Approach (FCA) and partially being used as a coupling framework within the GENIE application. BFG permits rules relating to compositions to be built and wrapper codes to be generated. Our work has similar goals and results to theirs but we make extensive use of metadata from which the composition rules are inferred. Our work is more suitable in a scalable environment where exact rules are difficult to specify, especially when nature of evolving components are not known in advance. Further, our metadata supports specification of rules where necessary but does not depend upon them for its operations. In addition to this, our work exploits the runtime correlated metadata to seek optimisation opportunities.

The THEMIS framework [10] demonstrates how component specific metadata can be used to perform cross component optimisations. Using component metadata for optimising resources and applications in the context of Grid is considered in [9]. Organisation of metadata for component-oriented compositions are discussed in [17]. Our work utilises some of their principles and techniques in advancing the solutions. We also make extensive use of some or part of the wrapping techniques outlined in [5, 12, 3]. However, our implementation of wrappers also function as part of the web-service so that the metadata can be published as part of the service.

OASIS4 [15] offers a component model description and configuration for coupled models. The framework is specifically aligned towards issues related to climate earth system modelling. Although it proposes earth-specific metadata, the metadata does not capture the details which are relevant to our work.

7 Conclusions

In this paper we have highlighted two issues that arise when performing composition of components, namely validity and efficiency. We also discussed the importance of addressing these two issues both in the context of legacy applications and modern component-oriented programming. We placed a particular emphasis on providing additional information related to components, which we call metadata, and using the same to automate or semi-automate the process of component composition.

Carefully specifying the metadata such that it captures the component and domain-specific information leads to potential benefits, which we outlined in Section 5. We used a legacy application as a driving example to illustrate the details and nature of the metadata for components based on legacy code. We also highlighted our plans in staging the metadata for validating the models and improving the efficiency of models.

However, we discussed only a subset of the issues arising in performing compositions of components. A number of interesting issues remain to be addressed.

- Currently, we do not capture or specify any information as part of the metadata to parallelise the simulation and to improve the performance. This issue is partly addressed and included in the BFG framework through rules. However, we intend to take a different approach where the parallelism can also be inferred from metadata if not specified explicitly.
- The experience-specific metadata may dramatically increase in size and may affect the overall performance. We plan to separate this aspect of the metadata in a separate consolidated storage (such as a common database).
- Manually specifying the metadata for legacy components is a complex process. Diminishing or ageing skills in legacy systems, unavailability of up-to-date documentation, willingness of new programmers to get a deeper understanding of the code are some of the contributing reasons for the difficulty in keeping pace with the changes. However, fortunately, GENIE is based on FORTRAN code-base and it is not entirely impossible to cope with

the ageing code exposure — if FORTRAN code is assumed to be not in parallel with modern programming languages.

- One of the interesting outcomes of using the metadata is how the application or component becomes free from value-specific logics. It is possible to take an aggressive approach here to migrate more issues into the metadata from the component/application. For instance, computational models of components may consist of or may have access to alternative solvers/smoothers. This can be captured inside the metadata. However, such an aggressive approach may increase the overheads in handling the metadata. There is an optimum amount of information that can be passed as metadata and rest in the application/component logic. It is interesting to observe the optimality and defining metrics for such optimality.
- As mentioned in Section 3, efficiency of a composition is a multivariate function. We simplified the model and assumed that performance is the primary concern. However, in a real setup, factors such as cost may also need to be considered.

At present, we are investigating means for unifying and generalising the metadata specification across different components and across the whole domain. We are also implementing the framework for extracting and staging the metadata in performing compositions. We are certain that the proposed approach could be used to solve associated issues in other domains of applications and in contemporary component-based systems.

References

- [1] Grid ENabled Integrated Earth system model (GENIE). <http://www.genie.ac.uk>.
- [2] Parallel Grid Runtime and Application Development Environment (PGRADE). <http://www.lpds.sztaki.hu/pgrade/>.
- [3] David M. Beazley and Peter S. Lomdahl. Controlling the data glut in large-scale molecular-dynamics simulations. *Comput. Phys.*, 11(3):230–238, 1997.
- [4] Olav Beckmann, Anthony J. Field, Gerard Gorman, Andrew Huff, Marc Hull, and Paul H. J. Kelly. Overcoming barriers to restructuring in a modular visualisation environment. In *LCR '04: Proceedings of the 7th workshop on Workshop on languages, compilers, and run-time support for scalable systems*, pages 1–7, New York, NY, USA, 2004. ACM Press.
- [5] Stephen H. Edwards. Toward reflective metadata wrappers for formally specified software components, 2001.
- [6] David Flanagan and Brett McLaughlin. *Java 1.5 Tiger: A Developer's Notebook*. O' Reilly & Associates, Inc., 2004.
- [7] Ian Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. In *IFIP International Conference on Network and Parallel Computing*, volume 3779 of *Lecture Notes in Computer Science*, pages 2–13. Springer, 2005.
- [8] J. Freire, C.T. Silva, S.P. Callahan, E. Santos, C.E. Scheidegger, and H.T. Vo. Managing Rapidly-Evolving Scientific Workflows. In *Proceedings of The 2006 International Provenance and Annotation Workshop*, page (to appear), 2006.
- [9] Nathalie Furmento, Anthony Mayer, Stephen McGough, Steven Newhouse, Tony Field, and John Darlington. Optimisation of component-based applications within a grid environment. In *Supercomputing '01: Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, pages 30–30, New York, NY, USA, 2001. ACM Press.
- [10] Paul H. J. Kelly, Olav Beckmann, Tony Field, and Scott B. Baden. THEMIS: Component Dependence Metadata in Adaptive Parallel Applications. *Parallel Processing Letters*, 11(4):455–470, December 2001.
- [11] Serge Lidin. *Inside Microsoft .NET IL Assembler*. Microsoft Press, 2002.

- [12] Alessandro Orso, Mary Jean Harrold, and David S. Rosenblum. Component Metadata for Software Engineering Tasks. In Wolfgang Emmerich and Stefan Tai, editors, *EDO*, volume 1999 of *Lecture Notes in Computer Science*, pages 129–144. Springer, 2000.
- [13] S. Panagiotidi, J. Cohen, J. Darlington, Marko Krznarić, and E. Katsiri. Service-enabling legacy applications for the genie project. In *All Hands Meeting, Nottingham*, September 2006.
- [14] M.K. Bane C.W. Armstrong R.W. Ford, G.D. Riley and T.L. Freeman. Gcf: A general coupling framework. *Concurrency and Computation: Practice and Experience*, 18.
- [15] R. Vogelsang D. Declat H. Ritzdorf S. Valcke, R. Redler and T. Schoenemeyer. Oasis4 user’s guide. *PRISM Report*, 2004.
- [16] Douglas Thain, Todd Tannenbaum, and Miron Livny. Condor and the grid. In Fran Berman, Geoffrey Fox, and Tony Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., December 2002.
- [17] Jeyarajan Thiyagalingm and Vladimir Getov. A Metadata Extracting Tool for Software Components in Grid Applications. In *Proceedings of the IEEE JVA Conference, 2006*, 2006.
- [18] Raja Vallée-Rai, Laurie Hendren, Vijay Sundareshan, Patrick Lam, Etienne Gagnon, and Phong Co. Soot - a Java Optimization Framework. In *Proceedings of CASCON 1999*, pages 125–135, 1999.