

Generation of an Integrated Development Environment (IDE) for Berkeley Open Infrastructure for Network Computing (BOINC)

Christian Benjamin Ries¹, Christian Schröder¹, and Vic Grout²

¹Computational Materials Science and Engineering
University of Applied Sciences Bielefeld, Germany
e-mails: [christian_benjamin.ries, christian.schroeder]@fh-bielefeld.de

²Centre for Applied Internet Research
Glyndŵr University, United Kingdom
e-mail: v.grout@glyndwr.ac.uk

Abstract

Model Driven Engineering (MDE) is a software development process based on domain specific models and is applicable in any computer engineering challenges which make processes more transparent to different users in the individual domains. The Unified Modeling Language (UML) can be used as a powerful tool within MDE to describe complex architectures which are more intelligible and precise. In this paper, we will discuss the first steps of a new MDE process - with help of the UML, Domain-specific modeling languages (DSML), and a generated Integrated Development Environment (IDE) - to offer the opportunity to model a whole Berkeley Open Infrastructure for Network Computing (BOINC) project with less time and maintenance effort. Within this MDE process we defined six diagram types, each for a unique purpose for a better separation of modeling aspects. These research and development effort lowers the barrier for entry to create an own Volunteer Computing project based on BOINC.

Keywords

BOINC, Model/View/Controller, Model/View/Delegate, Integrated Development Environment, UML, Code Generation

1. Introduction

Over the past five decades, software researchers and developers have been creating abstractions that help them program in terms of their design intent rather than the underlying computing environment, e.g. CPU, memory, and network devices are more understandable by use of Application-programming interfaces (APIs) (Schmidt, 2006). In this paper we use Berkeley Open Infrastructure for Network Computing (BOINC) to build up a world-wide computer cluster based on resources released by volunteers (Anderson *et.al.* 2006). We will define an abstraction layer on top of BOINC and encapsulate BOINC's architecture and functionalities to create a Model Driven Engineering (MDE) process. We will discuss how we can generate an

Integrated Development Environment (IDE) by use of a Unified Modeling Language (UML) Profile (OMG UML, 2011), three Domain-specific modeling language (DSML) specifications, and a template-based Code-Generation (CG) within our research project Visu@IGrid (VG). VG covers research towards an all-in-one IDE to model a whole BOINC project with the help of UML, code generation facilities, and separation of related functionalities to specific diagrams. Some effort to define and implement a MDE environment with support of textual and graphical elements is spent for tool-chains within Eclipse, e.g. Eclipse Modeling Framework (EMF), UML2Tools, Xtext (Eclipse, 2011). EMF is one prominent example but different in its way of modeling. EMF tends to take a bottom-up approach whereas UML tends to take a top-down approach (Gerber and Raymond, 2004). Additionally, EMF is the core component of the Eclipse Graphical Modeling Framework (GMF) and it has been shown that GMF lacks in support of evolution, and that the GMF infrastructure has a number of limitations, some of them related to co-evolution, i.e. changes in the EMF model would make the GMF editor unusable and must be changed in most parts (Ruscio, 2010). In our work we present an approach we will discuss how we could handle this gap by the use of a specific model specification by use of a UML Profile and DSML's to create an IDE with the minimum support to define a BP.

The remainder of this paper is organized as follows. Section 2 gives an overview of the VG, the problem domains, and how this paper uses well established architectural software implementation patterns. Next, Section 3 covers the main content of this paper and describes our approach to model a BOINC project (BP) with help of a UML Profile and our approach to generate an IDE for BOINC development. In Section 4 some open tasks and research questions are listed. Finally, Section 5 concludes this paper.

2. Visu@IGrid for BOINC

BOINC is a framework for solving large scale and complex computational problems by means of Public Resource Computing (PRC). BOINC is based on the principle of PRC where the computational effort is distributed in separated so-called workunits onto a large number of computers connected by the Internet. Volunteers provide their released computer resources, download progress workunits, and send back the result to one BP (Anderson, 2004). On the one hand, installing, configuring, and maintaining a BOINC based project is a highly sophisticated task (Ries *et.al.* 2010) and a lot of experience regarding the underlying communication and operating system technologies are needed for most application. On the other hand to handle these challenges, VG would fill these gaps.

VG is a research project with the goal to have specifications and definitions how to create a BOINC project within a Model Driven Engineering process. Furthermore, an IDE based on Code Generation (CG) facilities and an UML Profile (Ries *et.al.* 2011) should be created to have an abstraction of all BOINC functionalities. This abstraction makes it feasible to have a non error-prone process to create and run BP's with minimum work and time effort. Figure 1 shows one viewpoint of the underlying architecture of VG. In fact, VG is based on three different projects: (1) *Visu@IGridML*, (2) *Visu@IGridCG*, and (3) *Visu@IGridIDE*. Some work is done in

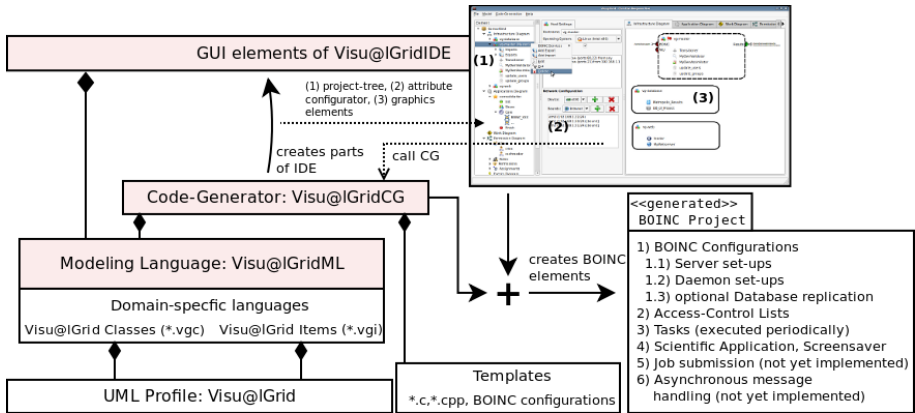


Figure 1: Workflow of an IDE generation, followed by a generation of a BOINC project based on the UML Profile Visu@IGrid, two DSMLs, and templates files. any project (Ries *et al.* 2010).

Visu@IGridML (VGML) includes DSML definitions and specifications. These values are twofold:

- (1) VGML defines a UML Profile to create models which can describe a whole BP with support to specify the architecture and functionalities of one BP, i.e. how many hosts are used, how should BOINC daemons work, which scientific application should be distributed and which kind of workunits should be deployed., and
- (2) VGML defines DSML's which are used by Visu@IGridCG to generate an IDE with support of an architectural tree-view, window forms to edit attributes, and a graphical view of a BP and associations between BOINC components, e.g. results of a computation should be stored on one specific host or in a database.

The IDE itself will be equipped with a module to optionally export a BP description which could be used by other code generators to create all necessary scripts, configuration files, and code implementations of a BP. Fig. 1 shows the first generated version of Visu@IGridIDE in the top-right area.

2.1. UML Profile: Visu@IGrid

UML Profile is a kind of UML extension mechanism. It specializes some of the language elements, imposes new restrictions on them while respecting the UML metamodel and leaving the original semantics of the UML elements unchanged. Icons and symbols can be specified and applied for these specialized elements. The Object Management Group (OMG) maintains some common and widely accepted profiles, such as SysML (OMG SysML, 2010). Stereotypes extend standard UML meta-classes. We have already specified a UML Profile called **UML Profile**

Visu@IGrid (UMLVG). UMLVG defines 50 stereotypes which are applicable to model a complete BP (Ries *et.al.* 2011). Current version of UMLVG defines no diagram types which could be used to encapsulate different focused models, e.g. a BP has an infrastructure based on one or more hosts and different sets of local or remote network-access restrictions for users with different owned rule-sets, e.g. permissions to administrate a whole BP or users with permissions just to request complete workunits.

2.2 Software Patterns: Model/View/Controller, Model/View/Delegate

One software pattern is the Model/View/Controller (MVC) architecture which is often used in Graphical User Interfaces (GUIs) to separate one data Model from graphical screen representations, so-called *View*. Between the *Model* and *View* is the *Controller* placed to handle the way the GUI reacts to user input. Requests by the *View* and changes of the *Model* should be synchronized to have an always up-to-date state. We already defined and implemented a DSML with an additional framework to enrich BOINC with a more handy and easier to use API (Ries *et.al.* 2010b). Our DSML approach makes it possible to decrease the lines of code for one fully executable BOINC application by approximately 77 percent based on a MVC architectural framework. In contrast, Qt uses a Model/View/Delegator (MVD) concept, where the MVC-Controller is replaced by a *Delegator* (Qt, 2011). The *Model* and *View* are still used. The *Model* class is responsible to offer and edit data items. The *View* renders items of the *Model*. How the items are rendered is defined by the *Delegator*. Between the *Model*, *View*, and *Delegator* one or more data items are exchanged and the underlying Qt framework keeps all views of one model synchronized. In our VGIDE we use more than one view of a model, i.e. Fig. 2 shows on the left-hand side an architectural tree view and on the right-hand side a graphical view of the same model. Unfortunately Qt's version 4.7 implementation does not support direct synchronization of an architectural tree model with a graphical representation. One work covers an approach to handle this gap (Schile, 2008) and this idea is partially used in our implementation.

3. Visu@IGrid Integrated Development Environment

UMLVG defines no diagrams; as a consequence some more effort is required to fill this gap. Here, main goals are:

- (1) the approach should be easy to apply by persons who are not familiar with UML, and
- (2) it should be possible to recreate the IDE at each time if it is necessary without lose of already done implementations, i.e. if descriptions of BOINC's validator or assimilator are done, they should be used in the new generated VGIDE.

Furthermore, the view of the hierarchy tree of one BP within the IDE should support context-sensitive menus to create or manipulate the structure of a BP and changes of the hierarchy should be synchronized by the underlying MVD implementation to have an always correct graphical representation of a BP, i.e. several levels of a hierarchy could only own associated elements, e.g. a host owns shares, an application

owns input files.

3.1. Idea

We created two new DSML specifications which are used to generate the structure of an IDE with support of contextual menus, automatic generation of forms to manipulate the attributes of UMLVG elements, synchronization of the underlying data structure with the hierarchy tree and graphical representation, and facilities to export the structure which could be used by external code generators. Fig. 2 shows part of these DSML's and the relation to our VGIDE. The DSML's have different file-extensions: (1) *VisualGrid Classes (*.vgc)*, and (2) *VisualGrid Items (*.vgi)*. VGC is used as an extension of the UMLVG to describe in which type of diagram the elements are shown, and VGI describes how the elements are visualized within the graphical view of VGIDE.

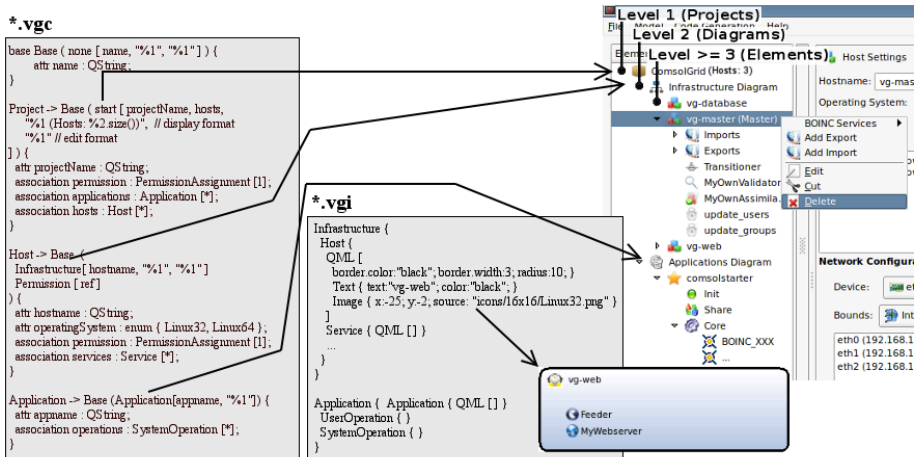


Figure 2: Conceptual idea and first realization of two DSMLs to generate a hierarchy tree and graphical representation of UMLVG elements.

VGC covers three important aspects:

- (1) Description how elements are related to each other; described by the notation "*class name* \rightarrow *class name of parent*" and by each element owned associations.
- (2) Description in which diagram the elements are shown (see Section 3.2). Elements could be added to one or more diagram, e.g. *Host* is added to the Infrastructure and Permission Diagram. Within the Permission Diagram *Host* is just a reference and could not be manipulated, but set of permissions could be associated with it, i.e. one user could be set-up to have permission to remote log-in on this host.
- (3) Description which information and how they are shown in the hierarchically tree. Used strings in the tree are described by format strings: "*%1 (Hosts: %2.size())*". During execution the placeholders - "*%1*" and "*%2*" - are

replaced by predefined variables, i.e. “*projectName*” and “*hosts*” in this case. The type of these variables are checked automatically by VGCG, i.e. “*projectName*” is a kind of string and could be used directly, “*hosts*” is an association and has a multiplicity “[*]” to keep zero-or-more elements with type of *Host* on which methods could be executed.

VGI is used for the representation of graphical items related to elements in the VGC and could be described by other DSML specifications, e.g. in this example we use Qt’s Modeling Language (QML, 2011) to specify a graphical node for one host. QML makes it possible to embed descriptions of graphical nodes within our DSML and to replace them during runtime.

3.2. Concept

As mentioned, UMLVG defines no types of diagrams, this is done by VGC. Here, we defined six diagrams for different purposes:

- **Infrastructure Diagram:** This diagram is used to specify the infrastructure of one BP. We can add different hosts, Storage Area Networks (SAN), BOINC services and periodically executed tasks. Fig. 1 includes an example of a generated IDE with a diagram of three hosts with few added BOINC Services (BS’s), i.e. BOINC daemons, databases, crontab tasks, webserver installation, and other applications. This diagram defines the relation between hosts and where computational results should be stored. The infrastructure diagram is a kind of UML Deployment diagram.
Applications Diagram: Within this diagram logical parts are specified, how an application would be executed or which third-party libraries are used. Furthermore, it is possible to add a wrapper application (Ries and Schröder, 2010b) which would make it feasible to use de-facto industry standard software components. This type of diagram is a kind of UML Statechart and Sequence diagram. The structure of this diagram could be more restricted, restrictions are depended by the type of application. On the one hand if a legacy-application like COMSOL Multiphysics (Ries *et.al.* 2010) should be used, it is not necessary to define a whole state or sequence graph, because the underlying wrapper-implementations handle the execution. On the other hand if an individual implementation is required, the structure of this BOINC application (BA) could be separated into four main sections: (1) **init**, (2) **share**, (3) **core**, and (4) **finish**, described in Table 2. Within the UMLVG these sections are defined by UML Classes which are associated to one or more UML Classifier.
- **Work Diagram:** This diagram is used to define the structure of workunits. We can specify which file aliases must be used within an Application Diagram and how the structure of input and output is. This type of diagram is a kind of UML Class diagram.
- **Permission Diagram:** Role-Based Access Lists (RBAL) are defined in this type of diagram. Within any users, roles and permissions could be added, cross-references are used to associate set of permissions to resources, i.e. users are not associated directly to resources like hosts, they are associated

to roles, these roles to set of permissions and finally to resources (Cirit and Buzluca, 2009).

- **Events Diagram:** BOINC could handle asynchronous communication to and from BOINC server to volunteers. The type of event could be defined in this diagram and the implementation could be referenced to elements within our mentioned Applications Diagram. This type of diagram is a kind of UML Sequence diagram. Unlimited events could be added and each is a start of a sequence to handle separated events, e.g. if one host becomes unreachable the BOINC services can be started on other hosts.
- **Timing Diagram:** As mentioned for Infrastructure Diagram, a BP could have periodically executed tasks. They are more specified in this Timing Diagram where we could add time-ranges for execution timing. This type of diagram is a kind of UML Class diagram.

Table 1 covers definitions of a static structure of our hierarchy tree model. The first three levels must be available in new BP's. It is not necessary that level 0 is shown. All leaves in this tree could be exported and external leaves are importable. Restrictions are present in the underlying UMLVG, i.e. exports are always possible and leaves of this tree which are imported must be valid on the point-of-interest.

Tree Level	Description
Level 0	First column of one tree describes the model root element and is hidden as default and owns project nodes.
Level 1	Describes separate BP's within one IDE. Fig. 2 shows one BP named " <i>ComsolGrid</i> ".
Level 2	Specifies default diagrams to support a full Model Driven Architecture (MDA) process to create a new BP.
Level >= 3	Each diagram could only stand for context relevant specifications, i.e. an infrastructure diagram specifies how a BP will be deployed. More hierarchically levels are used for contextual relevant elements and attributes, e.g. a host could contain ex-/imports and BOINC services. Each additional leaf hold dependent elements, e.g. a network interface card for users make no sense. Which elements are possible to add is defined by the UMLVG and VGC in Fig. 2. Only elements which are defined for one diagram are possible to add.

Table 1: Columns definition of a Visu@IGrid Tree Model

Our IDE is based on a clear structure matched with COMSOL Multiphysics (COMSOL, 2010). Figure 1 shows on the top-right hand-side our IDE. Three areas with well-defined purposes are shown: (1) the column of the left-hand side shows our hierarchy tree based on UMLVG, (2) the column in the middle shows contextual sensitive attributes/settings (AS) for selected elements within the tree, and (3) last column on the right-hand side shows our graphical representation of the hierarchy equal to or higher than level 2. AS's are based on our predefined UMLVG

stereotypes. They could be extended if higher fine-grained AS's are needed. The form to manipulate element attributes is generated by the VGCG and widgets are selected on the data type, e.g. string values are editable by text fields, associations are editable by selection lists.

Application Sections	Description
Init	Bas could be initialized in different ways, this depends on the target processor (e.g. multicore processors or Graphics Processing Units) and some special diagnostics features must be used, e.g. for debugging purposes.
Share	In any circumstances where it is needed to exchange values between more than one executed application, it can be defined which values are exchanged and how the data is retrieved by one application, i.e. an application have to set values for variables to exchange with external applications.
Core	Predefined BOINC functionalities (included in BOINC's framework) could be added here or own implementations are addable, i.e. textual (DSL, C/C++-Code), graphical (UML) implementations. Some work is done here, we already defined a way of a MVC concept and how Aspect-Oriented Programming (AOP) can be used (Ries <i>et.al.</i> 2010a).
Finish	Clean-up the execution environment, i.e. closing open files or database connections.

Table 2: Predefined section for the application implementation.

3.3. Benefit

With this code-generation and IDE facilities we have tool support for a wide range of different field of application. Different DSML specifications allow creation of IDE's for specific areas. This approach fills the gap between lightweight tools and heavy tools like Eclipse Modeling Framework which are not practicable in projects with a lot of changes of the underlying model from one day to another (Ruscio, 2010). It is only necessary to change the code-generation process to create executable projects. This work enables us to work further within the research project Visu@IGrid and it is a good starting point to work toward a code generator for full support to generate a whole BP.

4. Future Work

Future work will be focused on the implementation of a parser for the Object-constraint language (OMG OCL, 2011) to set-up rules between associations of UMLVG elements.

Additional work has to be done on the underlying architecture and how models can be converted into executable code, i.e. a modeled BP can be exported to XML files

and this is the end of the workflow currently. We have to work toward code generators to transform these XML files into code of scientific applications, validator and assimilator for computational results and tools for a practicable maintaining of a BP.

5. Conclusion

In this paper we have presented an approach to extend an already defined UML Profile for BOINC by Domain-specific modeling languages to make it easier and more intelligible to define and understand available BP's. We mentioned two Domain-specific modeling languages and added six diagram types with special purposes to specify BOINC's infrastructure, applications, how workunits must be used, who has permissions for different areas, how asynchronous communication is handled, and when selected tasks should be executed. Furthermore, we have defined a first structure of an IDE for Visu@IGrid which is automated generated, where each area has a well-defined purpose. With this effort the barrier to work with BOINC is lowered. The maintaining effort of one BOINC Project is less, because it is only necessary to manipulate the model within VGIDE, to export it to XML and then to create a BOINC Project by a not currently available code generator.

6. Acknowledgement

This work has been fully funded by the German Federal Ministry of Education and Research.

7. References

Anderson, D.P., Christensen, C., and Allen, B. (2006), Designing a Runtime System for Volunteer Computing, UC Berkeley Space Sciences Laboratory, Dept. of Physics, University of Oxford and Physics Dept., University of Wisconsin – Milwaukee

Anderson, D.P. (2004) BOINC: A System for Public-Resource Computing and Storage, *5th IEEE/ACM International Workshop on Grid Computing*

Cirit, C. and Buzluca, F. (2009), A UML Profile for Role-Based Access Control, *ACM SIN*

COMSOL Multiphysics Press Release Web Site (2010), "COMSOL Multiphysics Version 4.0 ab sofort verfügbar", <http://www.comsol.de/press/news/article/648/>, (Accessed 28 June 2011)

Eclipse Modeling Tools (2011), <http://www.eclipse.org/home/categories/index.php?category=modeling>, (Accessed 20 August 2011)

Gerber, A. and Raymond, K. (2004), MOF to EMF: There and Back Again, *Cooperative Research Center for Enterprise Distributed Systems (DSTC)*, University of Queensland, Brisbane, Australia

Object Management Group (2011), OMG Unified Modeling Language (OMG UML), Superstructure, <http://www.omg.org/spec/UML/2.3/Superstructure/>

Object Management Group (2011), OMG Object Constraint Language (OMG OCL), <http://www.omg.org/spec/OCL/2.0/>

Object Management Group (2010), OMG Systems Modeling Language (OMG SysML), <http://www.omg.org/spec/SysML>

Qt Reference Documentation (Version 4.7), <http://doc.qt.nokia.com/latest/model-view-programming.html>, (Accessed 28 June 2011)

Qt QML (Version 4.7), Introduction to the QML Language, <http://doc.qt.nokia.com/4.7-snapshot/qdeclarativeintroduction.html> (Accessed 22 August 2011)

Ries, C.B., Schröder, C., and Grout V. (2011), UML Profile for Berkeley Open Infrastructure for Network Computing (BOINC), submitted to *IEEE Conference on Computer Applications & Industrial Electronics (ICCAIE)*

Ries, C.B., Hilbig, T., and Schröder, C. (2010a), A Modeling Language Approach for the Abstraction of the Berkeley Open Infrastructure for Network Computing (BOINC) Framework, *Proceedings of the International Multiconference on Computer Science and Information Technology (IMCSIT)*

Ries, C.B. and Schröder, C. (2010b), ComsolGrid - A framework for performing large-scale parameter studies using Comsol Multiphysics and Berkeley Open Infrastructure for Network Computing (BOINC), *Proceedings of the COMSOL Conference*

Ruscio, D.D., Lämmel, R., Pierantonio, A. (2010) Automated co-evolution of GMF editor models, *CoRR*, Volume abs/1006.5761, June 2010

Schiele, J. (2008), Being Qt in Theoretical Computer Science - Design and implementation of a program for editing and viewing of finite automates, *Studienarbeit*

Schmidt, Douglas C. (2006), Model-Driven Engineering, *IEEE Computer Society*, Volume 39, Number 2, February 2006, pp25-31