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

Christian Benjamin Ries
Christian Schröder, and Vic Grout

E-Mail: cbr@fh-bielefeld.de
Website: www.visualgrid.org

Seventh Collaborative Research Symposium on Security, E-learning, Internet and Networking, Furtwangen (Germany), 25th October 2011
Outline

• BOINC – what is this? 🌟
• Visu@lGrid – the basic idea!
• Conclusion & future work…
BOINC - Public Resource Computing
BOINC - Public Resource Computing
BOINC - Public Resource Computing

- [1] **Scientific Application** (Independent Software Vendor, Legacy-applications, personal creative implementations...)
- [1..*] runtime-parameter sets
- [*] additional files
BOINC architecture - overview

- min. 7 components
- others are optionally, and more could be added
BOINC architecture - overview

- any BOINC component could be started on different hosts
- main host “vg-challenge-01” needs access to any host
- all hosts needs access to projects database
BOINC work life-time

Long and hard way to survive!
Could be modified, own implementations are possible.
BOINC – what can we handle?
BOINC – what can we handle?

Out of the box

BOINC ready!

Not covered!
BOINC...

...can be used in different scenarios:

- native application [we have to do everything manually],

- native application [but multithreaded, a little bit more work has to be done for thread handling],

- on different platforms and architectures,

- single-/multi-core, GPU,

- legacy applications,

- synchronous, asynchronous messages,

- ...some more...
**BOINC**

*All scenarios need experience in one or more technology fields!*

Currently BOINC uses following technologies (could be extended):

- C/C++ programming language,
- Python programming language,
- BASH & CSH shell scripts,
- PHP for some monitoring, maintenance, and website elements,
- Perl (e.g. rBOINC),
- SQL (database queries),
- XML (configuration and RPC-requests),
- OpenCL & CUDA,
- …some more!
Where are we currently?

1st Programming Languages: Assembly, Fortran, C/C++

Hardwork: Bits-Bytes

Sequential, Threads, Parallel, Single-/Multi-Core, GPU, µC

(Distributed, Volunteer, Grid, Cloud) - Computing
Model-driven engineering (MDE) is a software development methodology [...] creating/exploiting domain models [...]. The MDE approach is meant to increase productivity by maximizing compatibility between systems

- via reuse of standardized models,
- via models of recurring design patterns in the application domain,
- via a standardization of the terminology, and
- the best practices used in the application domain.

MDE is considered effective, if its models make sense from the POV of a user that is familiar with the domain, and if they can serve as a basis for implementing systems. [...]

Visu@lGrid

Main goal: ONE MODEL -> ANY TARGET!

- Code-Generation (CG) will produce always valid and executable code!
- Platform-independent (PIM) and Platform-specific (PSM) models are definable, depends on CG!

![Diagram showing different platforms and frameworks related to BOINC, Visual Grid, and Textual Modeling](image)

Visu@lGrid

GUI elements of Visu@lGridIDE

(1) project-tree, (2) attribute configurator, (3) graphics elements

creates parts of IDE

call CG

Code-Generator: Visu@lGridCG

Modeling Language: Visu@lGridML

Domain-specific languages
Visu@lGrid Classes (*.vgc) Visu@lGrid Items (*.vgi)

UML Profile: Visu@lGrid

<<generated>>
BOINC Project

1) BOINC Configurations
   1.1) Server set-ups
   1.2) Daemon set-ups
   1.3) optional Database replication
2) Access-Control Lists
3) Tasks (executed periodically)
4) Scientific Application, Screensaver
5) Job submission (not yet implemented)
6) Asynchronous message handling (not yet implemented)
(1) Visu@lGrid

We created an UML Profile:

(1) Visu@lGrid

This UML Profile could be applied to UML diagrams:
(2) Visu@lGrid

We created some Domain-specific languages, e.g.

```plaintext
base Element {
    attr name : QString;
}

Project -> Element (name, projectName, "%1 (%2)") {
    attr projectName : QString { description "Name of the project, [... ]."; }
    attr workingDirectory : QString { description "Defines the directory where the project will be stored on all hosts."; }
    attr state : bool { description "Enables/Disables the execution of this project. Be aware of a wrong value!"; }

    // Diagrams
    association infrastructure : ptr Infrastructure [1];
    association applications : ptr Applications [1];
    association work : ptr Work [1];
    association timing : ptr Timing [1];
}

Host -> Element (hostname, "%1", "%1") {
    attr hostname : QString { def "vg-host-01"; }
    attr main : bool { def "false"; }
    attr operatingSystem : enum { Linux32, Linux64 } {
        def "Linux32";
        description "Describes the operating system which should be used on one host.";
    }

    // Start Services
    association feeder : Feeder [*];
    association transitioner : Transitioner [*];
    association assimilator : Assimilator [*];
    // END Services
    association nics : NetworkInterfaceCard [*];
    association exports : PortExport [*];
    association imports : PortImport [*];
}
```

Different types of diagrams, see (3). Could be changed for any project, not all are types of diagrams are necessary any time, depends on POV.
(2) Visu@lGrid

We created some Domain-specific languages, e.g.

Distributions {
    //serverLinux32: "~/ubuntu-10.1-i386.iso"
    serverLinux64: "~/ubuntu-10.1amd64.iso"
}

Project {
    maintainer: "Christian Benjamin Ries"
    contact: "cbr@fh-bielefeld.de"
    name: "visualgrid"
}

Host "vg-challenge-01" {
    services "feeder, ...");
    assimilator save on "vg-challenge-02" in "db.res"
    ...
}

Deployment

Syntax depends on (4).

Science App.

worker Spinhenge {
    cpp {
        int a = 42;
        if(a > 42)
            modeledFunction(a);
        else a = 42;
    }
}

worker Spinhenge {
}
(3) Visu@lGrid

Live demonstration...
(3) Visu@IGrid

Live demonstration...
(3) Visu@lGrid

Types of diagrams, we suggest 6 types:

- **Infrastructure**: add information of hosts, SANs, BOINC services (tasks, daemons), which scientific applications should be used, define where computational results should be stored.
(3) Visu@lGrid

Types of diagrams, we suggest 6 types:

- **Applications**: implementation of a scientific application, e.g. own creative implementation or a legacy-application (use with a wrapper), how to handle errors during execution, all logical algorithms are added here (*open task, currently hard-coded!*)

- **Work**: how parameter sets should look like, format of input and output files, definition of names to open them within a scientific application (*open task, currently hard-coded!*
(3) Visu@lGrid

Types of diagrams, we suggest 6 types:

- **Permission**: Role-Based Access Lists (RBAL), rule-sets for users

```plaintext
<<User>> Administrator
<<Permission>> DBFull
<<Host>> vg-database
<<Resource>> Metropolis_Results
<<Table>> SpinValues
<<Resource>> DB_of_Project
<<Role>> DatabaseAdmin
<<Role>> WebAdmin
<<Permission>>
<<Host>> vg-web
<<Webserver>> MyWebserver
<<Feeder>> feeder
<<Task>> $ update_users
<<Task>> $ update_groups
<<Application>> consoIstarter
```
(3) Visu@lGrid

Types of diagrams, we suggest 6 types:

➔ **Events**: kind of sequence diagrams *(open task, currently hard-coded!)*

➔ **Timing**: a time-line with some added tasks which are executed, tasks are defined in “Application” *(open task, currently hard-coded!)*
(4) Visu@lGrid

Planned:

_testbed_diagram_

Store in XML, and transfer to Visu@lGrid Controller.
Conclusion

- More implementation than research today.

- Some barriers are conquered to have a development environment to support our team to work more on research questions.

- Currently, we can deploy a new BOINC project, on an easy way.
Future work

- We have to work in more detail to get diagram specifications for
  - an Application diagram, with support for all BOINC life-time steps, e.g. work generator, validator, assimilator
  - a Work diagram
  - an Events diagram
  - a Timing diagram, we to check the possibilities of SysML
- A fully UML Profile specification for BOINC.
Thank you for your attention!

Learn more about SpinHerge@home and become part of it!
http://spin.fh-bielefeld.de

http://www.visualgrid.de
Thank you,


Telefon: +49 (0) 521 106 71222
e-Mail: cbr@fh-bielefeld.de
Website: www.visualgrid.org

Computational Materials Science & Engineering (CMSE)
Room 202, Werner-Bock-Straße 36
33602 D-Bielefeld