Model-based Generation of Workunits, Computation Sequences, Series and Service Interfaces for BOINC based Projects

Christian Benjamin Ries
Christian Schröder, and Vic Grout

E-Mail: mail@christianbenjaminries.de
Website: www.visualgrid.org

WORLDCOMP12, SERP'12
Las Vegas (USA), 17\textsuperscript{th} July 2012
BOINC is a Public Resource Computing middleware framework

seq1: Work request

seq2: workunit (WU)

seq3: result

Computer 1

Computer N

Company-/Institute network

Participants!

Compute!
BOINC's Workunits (WUs)

Scientific Application
(Independent Software Vendor, Legacy-applications, personal creative implementations...)

[1..*] various sets of runtime-parameter
[*] additional files
Excerpt of BOINC's WU Lifetime

WU creation

WU performance

- Is done by specific scripts or other applications.

WU validation

- Is done by BOINC's default tools or own implementations.

WU assimilation

- Is done by an scientific application.

Any part which access WUs need specific interfaces. Furthermore, the validation and assimilation process can vary for any WU.

This transition is on the next slide.
Handling of failed WUs

Some steps are hidden.

Create new one and perform it.
Handling of WU Series

WU handling, id: 10

Serie 1: 

WU handling, id: 100

Serie 2: 

Serie 3: 

no

Serie 1 & 2 ready?

yes

WU handling, id: 110

Uses the result values of WUs (id: 10 and id: 100).
The Core Questions

➔ Which UML elements are necessary to create a model for WU creation?

➔ How can we model a sequential queue for WU progressing?

➔ How can BOINC’s validator and assimilator access result’s data on a higher abstraction level? In addition, is it possible to have only one interface or description which makes it possible to allow access by all BOINC components?

➔ How can we track the lifetime of WUs when they are used in different scenarios, e.g. one WU is used within a sequential performed queue?
UML Abstraction: Ranges

Example (excerpt of the case study):

Any LMBoincRangeRule::getValue() {
    std::string seriesName;
    for(auto s : series()) {
        if(s->activated()) {
            seriesName = s->name(); break;
        }
    }
    if(seriesName == "S1") datafield()->setData("edge");
    else if(seriesName == "S2") datafield()->setData("normalize");
    else if(seriesName == "S3") datafield()->setData("oil");
    else if(seriesName == "S4") datafield()->setData("negate");
    else if(seriesName == "S5") datafield()->setData("combine");
    return datafield()->data();
}
UML Abstraction: In-/Output, Datafields

- file type can be queried during CG
- file format can be determined during CG
- «InterfaceAssimilate/Validate» can be used to open result files
UML Abstraction: WUs on Demand

Different runtime Scenarios:

- **Static Series**
  - Each workunit is executed once.
  - Fixed set of workunits.

- **Continuous Series**
  - Workunits are continuously generated.
  - Dynamic addition and removal of workunits.

- **Dynamic Series**
  - Workunits are dynamically scheduled.
  - Flexibility in execution order and timing.

**Workunit**
- +id: Integer
- +name: String
- +rsc_fops_est: Float
- +rsc_fops_bound: Float
- +rsc_memory_bound: Float
- +delay_bound: Float
- +min_quorum: Integer = 2
- +target_results: Integer
- +max_error_results: Integer
- +max_total_results: Integer
- +max_success_results: Integer

**InterfaceDataSource**
- +getPath(): String

**Input**
- +name: String
- +copyfile: Boolean = false
- +sticky: Boolean = false
- +nodelete: Boolean = true
- +report_on_rpc: Boolean = false
- +unique: Boolean = false

**Range**
- +getValue(): Any
- +ranges: 0..*

**Series**
- +name: String
- +deadline: Integer = -1
- +activated: Boolean = false

**Application**
- 1
- 0..*

**InterfaceWorkunit**
- +create(): Boolean
- +cancel(): Boolean
- +information(): String
Handling of Series

- **Static**: All WUs are created before a «Series» will be created. Only these known WUs are handled by a BOINC project.

- **Continuous**: No WUs at the beginning of a «Series». WUs are created on demand, e.g. when new data packages are available or when a time slot is reached.

- **Dynamic**: The previous two possibilities are merged.
Series & WU States

First part of the UML statemachine diagram to monitor instances of «Workunit» and «Series». State’s top region is responsible for WU monitoring and creates new WUs when data is available or next WU in a sequence has to be performed. The bottom region monitors a «Series» and handles canceling events for a «Series» instance or if its finished and results can be merged.

Second part of the UML statemachine diagram. During computation of one WU, clients can decide to cancel current WU, and therefore it has a changed WU state. When it is finished it will be validated, if this validation failed the exit pseudostate is used. The followed assimilation state can decide to retry this WU and a new WU is created with same «Input» values. If this WU is in a sequence, “Next” is used otherwise the statemachine is finished.
Case Study

5 Series:

Series 1 (normalize)
Series 2 (painting)
Series 3 (negate)
Series 4 (edge)

⇒ Series 5 (merge)

5 Series with different converting modes:
Mode 1 -> normalize algorithm
Mode 2 -> painting algorithm
Mode 3 -> negate algorithm
Mode 4 -> edge algorithm
Mode 5 -> merge (1)-(4)

Pool of available workunits, ready for computation.
Future Work

- WU’s performance can have restrictions, e.g. the use of floating-point operations or allocation of hard disk space can be limited. Is it possible or reasonable to use the UML for this kind of specification?

- The current case study is hard-coded and cannot be changed during runtime. This can be changed by the use of a Domain-specific language (DSL) or by the use of a graphical language.

- Additional thought should also be spent on how the presented model can be used for conventional supercomputers, where other technologies like Message Parsing Interface (MPI) or OpenMP are used.
Thank you,

Christian Benjamin Ries

e-mail: mail@christianbenjaminries.de
website: www.visualgrid.org