Visu@lGrid - Realization of a development environment for model-based design and code generation of heterogeneous client server applications
• Visu@lGrid uses BOINC
  (Berkeley Open Infrastructure for Network Computing)

• BOINC is a Public Resource Computing infrastructure...
• ...can also be used in companies or in private networks.

Public Resource Computing (PRC) != Grid Computing (GC)

PRC:
… participants share their processor and computer resources
… asymmetric relationship between projects and participants
… projects are typically small academic research groups with limited computer expertise
… participants are individuals with different operating systems

GC:
… involves organizationally-owned resources: supercomputers, clusters, and PC's owned by universities, research labs, and companies
… resources are centrally managed by IT professionals and are powered most of the time
BOINC Architecture

Client side

Core Client

Server side

Data Server

Scheduling Server

Web Interfaces

Project Backend

Backend Application Server

BOINC DB

Project specific application component

BOINC component

Project specific client code

BOINC client API
set up database

set up webserver

set up BOINC

change BOINC permissions

change user restrictions

create configurations

...a lot of more to do!

program some specific code

add this applications to the project

add important tools!

- for different platforms!
- for different processors!
- for different memory amount!

- independent of the language!

write transitioner

write validator

write assimilator

write file deleter

create datafiles / workunits

start / stop
set up database

set up webserver

set up BOINC

change BOINC permissions

change user restrictions

create configurations

...a lot of more to do!

program some specific code

add this applications to the project

add important tools!

- for different platforms!
- for different processors!
- for different memory amount!

- independent of the language!

write transitioner

write validator

write assimilator

write file deleter

create datafiles / workunits

start / stop
XML/plain text/binary data
Processor targets

- Central Processor Units (CPU)
  - Single core
  - Multi-Core, supported by e.g. OpenMP, MPI

- Graphics Processing unit (GPU)
  - Open Computing Language (OpenCL)
Platforms / Architectures

- Windows, e.q. 32/64 Bit
- Linux, e.q. x86, SMP
- Mac OS/X

- ...Fieldprogrammable gate arrays (FPGA)?
- ...Playstation 3?
- ...home devices?
The Challenge...

... make it as easy as possible!
The Challenge...

... make it as easy as possible!

The main goal of this project is the realization of a software development environment called „Visu@lGrid“ which allows one to develop applications based on the „Berkeley Open Infrastructure for Network Computing (BOINC)“ by graphical and textual modeling and complete code generation!
Visu@IGrid – Development-Environment for modelling of graphical heterogeneous Client-Server-Application with automated Code-Generation

Modelling Language
Visu@IGridML

Diagrameditors
User Interface
Visu@IGridIDE

Code-Generator
Visu@IGridCG

Standardservices and Standardprotocols, standardized, open and general usable Interfaces (BOINC, Eclipse, UML, XML, etc.)
Visu@lGrid Modeling Language
Domain Specific Language

```plaintext
inc BoincIncludes {
  "~/boincadm/src"
  "~/boincadm/src/api"
  "~/boincadm/src/lib"
}
lib BoincLibraries {
  "~/boincadm/src/api", "boinc_api"
  "~/boincadm/src/lib", "boinc"
  "/lib", "pthread"
  "/usr/lib", "jpeg"
}
mode single;
processor cpu;
on heapcheck | memoryleakcheck;

In order to create applications with multicore or GPU computing support the following statement can be used:

mode multi;
processor gpu;
openc1 yes;
```
Domain Specific Language

**Calculation, simulation datasets and results:**

```plaintext
infile "metropolis_data.xml"
    as ObjectName1;
infile "param.jj" as ObjectName2;
infile "param.nn" as ObjectName3;
infile "param.ww" as ObjectName4;

After the execution of the client application the results are stored in *result files* defined by the statement `outfile` and are uploaded to the server.

```plaintext
outfile "metropolis_out.erg"
    as ObjectResult1;
```
Domain Specific Language

CALL CALLCULATION OR SIMULATION ROUTINES:

```plaintext
worker Spinhenge {
    use "do_work();"
}

This statement could be replaced by other instructions, e.g.

worker Spinhenge {
    cpp {
        int a = 42;
    }
    action(modeledFunction(a));
}
Domain Specific Language

LEGACY APPLICATIONS, WRAPPER ROUTINES:

```c
worker Spinhenge {
    wrapper {
            weight, checkpoint_filename,
            fraction_done_filename, ...])
    }
}
```
Thanks for your attention!

References:
