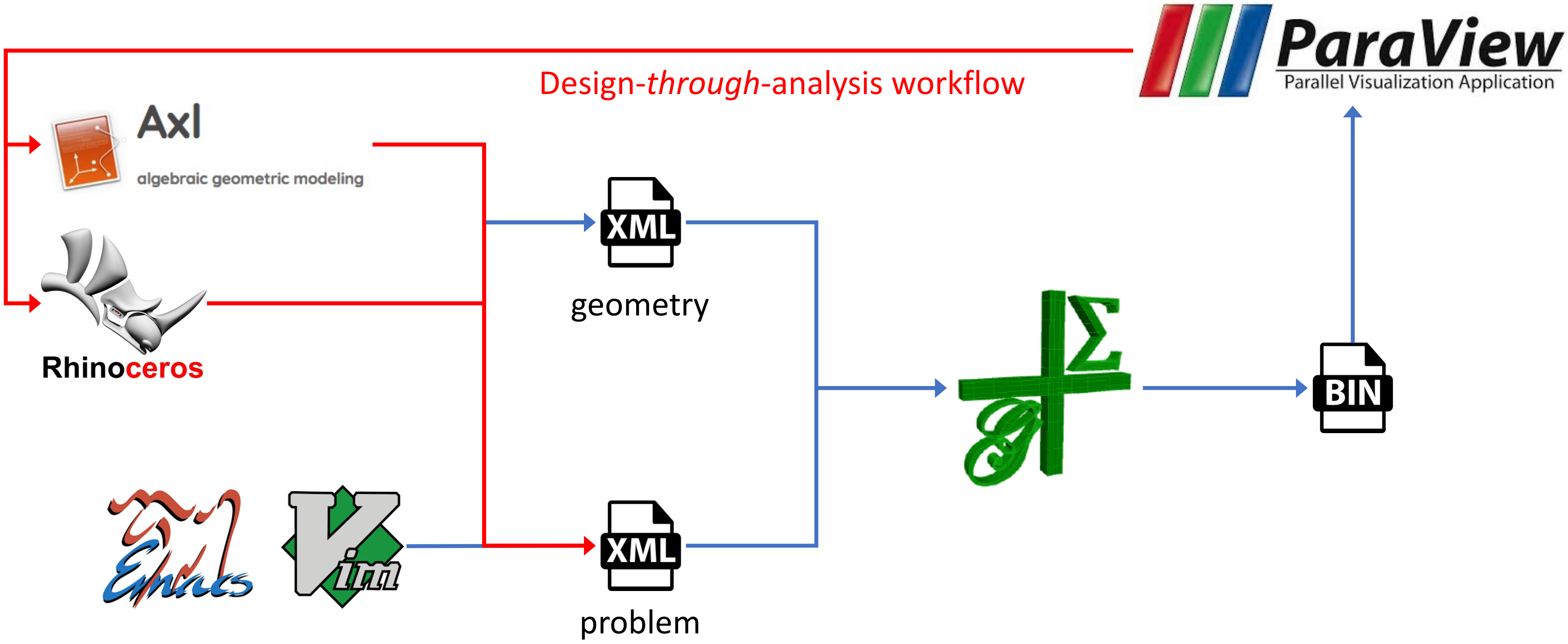
G+Smo interactive

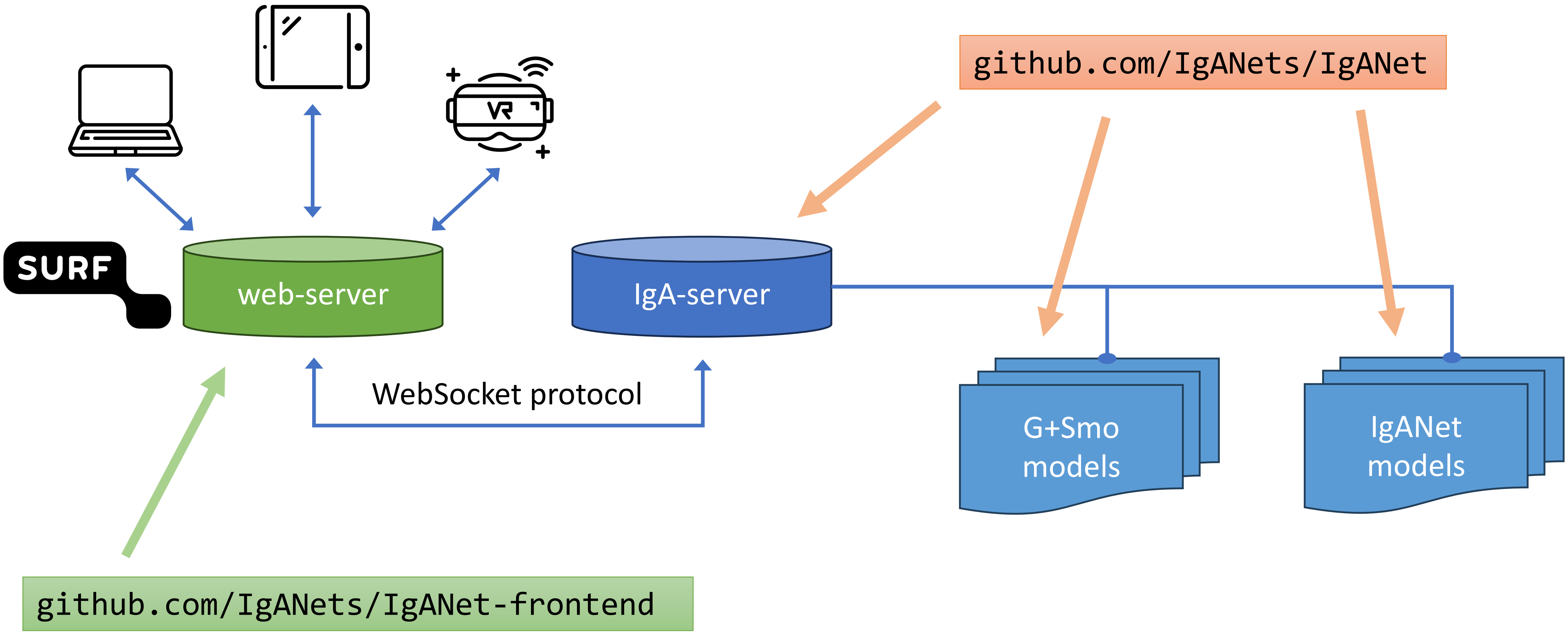
Matthias Möller (TU Delft)

G+Smo Developer Days, March 4th-6th, 2024, Thessaloniki, Greece

Current G+Smo workflow


ParaView
Parallel Visualization Application
Design-*through*-analysis workflow
Axl
algebraic geometric modeling
Rhinoceros
XML
geometry
XML
problem
BIN

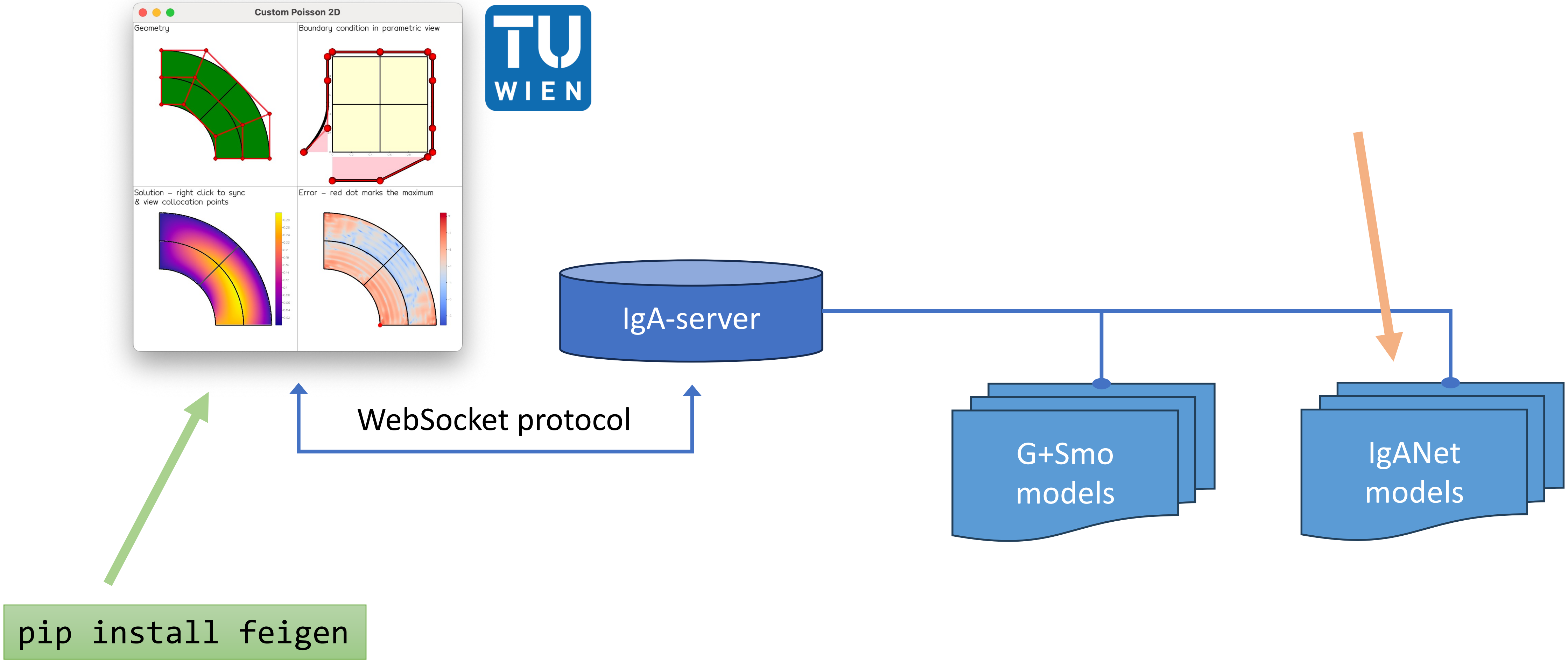
Collaborative DTA workflow



github.com/IgANets/IgANet

github.com/IgANets/IgANet-frontend

Collaborative DTA workflow

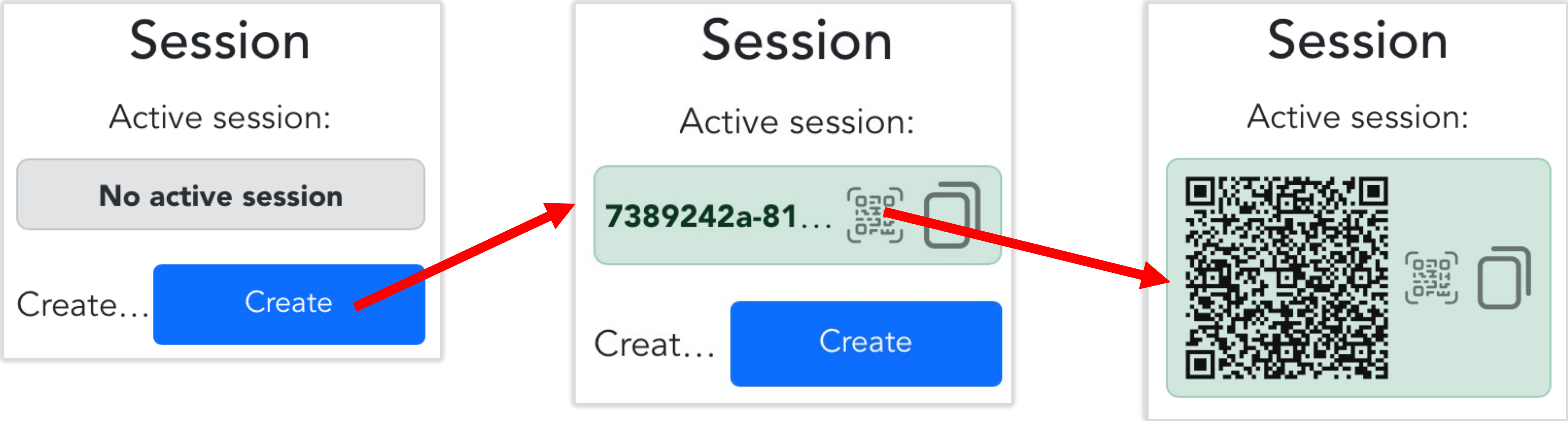


Custom Poisson 2D
Geometry
Boundary condition in parametric view
Solution – right click to sync & view collocation points
Error – red dot marks the maximum
TU WIEN
IgA-server
WebSocket protocol
G+Smo models
IgANet models

```
pip install feigen
```

Try it yourself

- Open in your browser https://visualization.surf.nl/iganet
- Create a new session or connect to your neighbor's session



Session

Active session: No active session

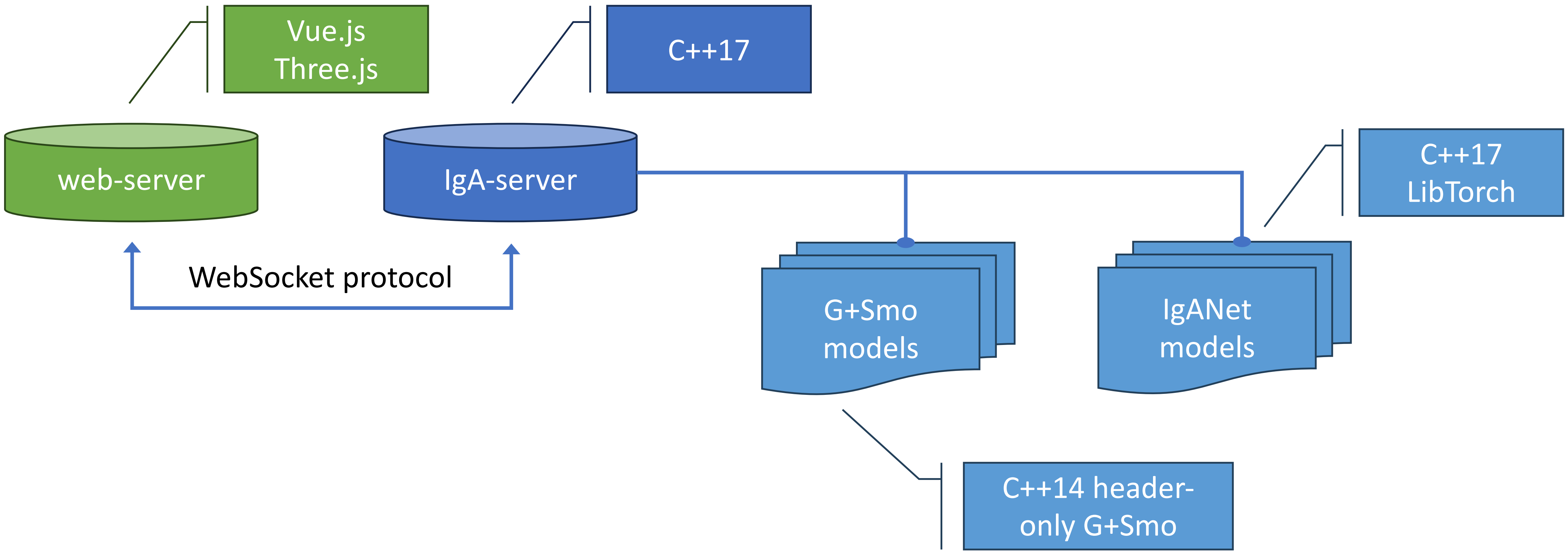
Session

Active session: 7389242a-81...

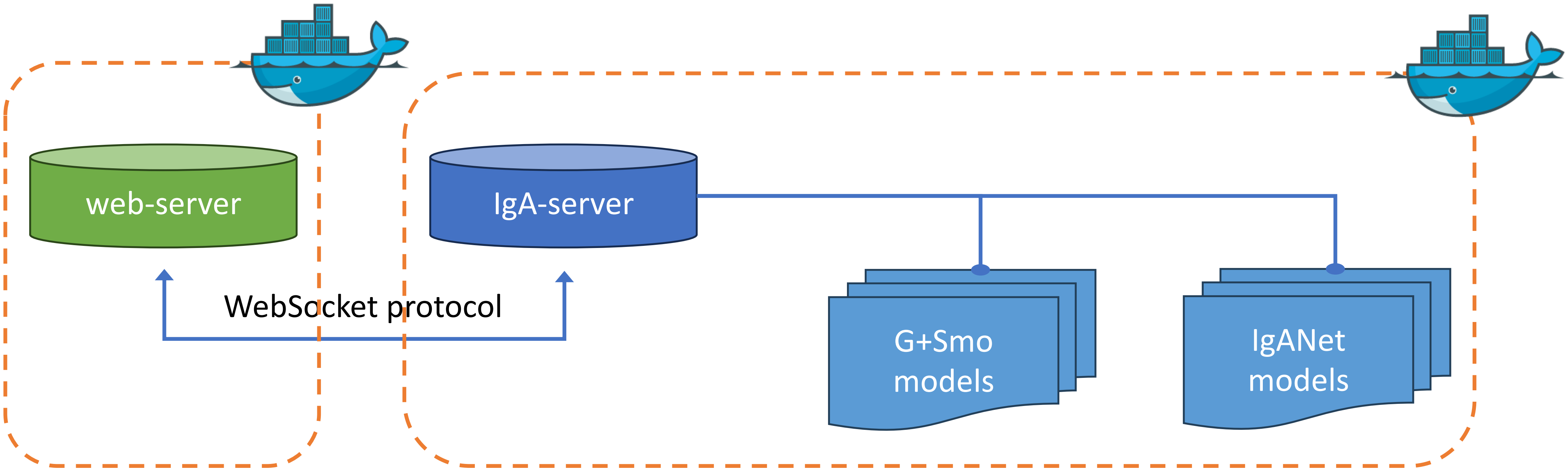
Session

Active session:

Software design



Software design



Plans to run model instances on-demand on Snellius cluster

Code organization (to be refactored)

```
webapps/
+-server.cxx
+-config/
  +-server.cfg
```

Don't care about this

```
+-models/
  +-<IgANet models>
  +-gismo/
    +-GismoModel.hpp
    +-GismoPdeModel.hpp
    +-GismoPoissonModel.hpp
    +-GismoPoisson2d.cxx
```

This is our playground for implementing new G+Smo models

The server

Multi-threaded server

```
./webapps/server -t #threads –m modelpath -p port –c cfgfile
```

- Loads all models (=shared libraries) in the modelpath
- JIT-compiles instances of templated modules and caches them
- Supports multiple independent (collaborative) sessions

The model's main file

`webapps/models/gismo/GismoPoisson2d.cxx`

- Implements the **create**-function (some models have a **load**-function)

```
std::shared_ptr<iganet::Model> create(const nlohmann::json &json)
{
    // extract degrees, ncoeffs, npatches from JSON

    return std::make_shared<
        iganet::webapp::GismoPoissonModel<2, iganet::real_t>
        >(degrees, ncoeffs, npatches);
}
```

GismoPoissonModel

```
template <short_t d  /* parametric dimension  */ ,
          class T    /* real-valued data type */ >
class GismoPoissonModel
    : public GismoPdeModel<d, T>, /* base model for all G+Smo PDEs */
      public ModelEval,           /* model supports 'eval' function */
      public ModelParameters      /* model has run-time parameters  */
{

nlohmann::json eval(const std::string &component,
                    const nlohmann::json &json) const override { … }

nlohmann::json getParameters() const override { … }

};
```

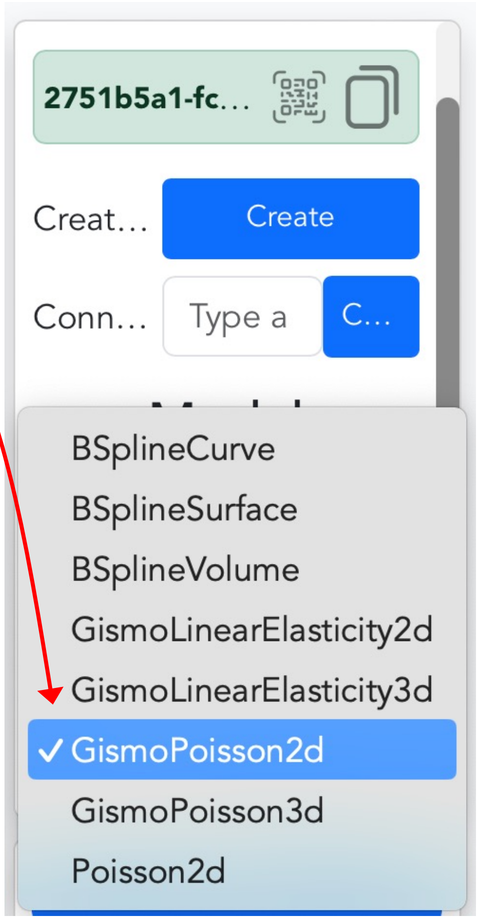
GismoPdeModel

```
template <short_t d, class T>
class GismoPdeModel
    : public GismoModel<T>,       /* basic stuff */

      public ModelElevate,        /* indicates that model supports  */
      public ModelIncrease,       /* degree elevation/increase and  */
      public ModelRefine,         /* refinement -> abstract methods */

      public ModelReparameterize  /* supports Ye's reparametrization */
{
protected:
      gsMultiPatch<T> geo_;       /* G+Smo multi-patch */
};
```

What you have to implement – bare minimum

- Functions from abstract base class(es)

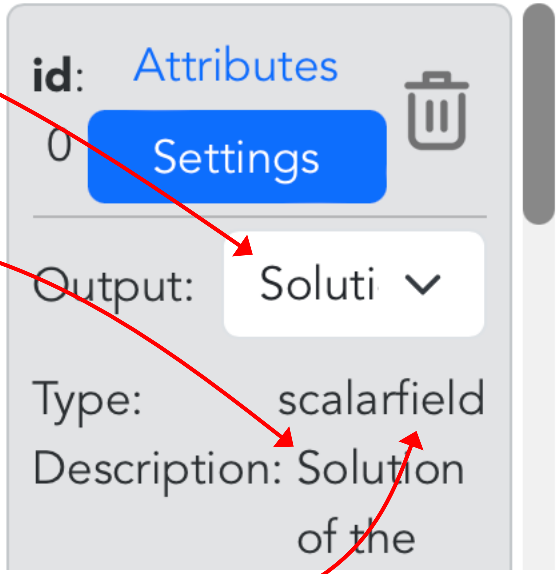
```
std::string getName() const override
{ return "GismoPoisson" +
  std::to_string(d) + "d"; }

std::string getDescription() const override
{ return "G+Smo Poisson model in " +
  std::to_string(d) + " dimensions"; };
```



What you have to implement – bare minimum

- Functions from abstract base class(es), cont'd

```
nlohmann::json getOutputs() const override
{
  return R"([{
    "name" : "Solution",
    "description" : "Solution of the …",
    "type" : 1}])"_json;
}
```



id: 0

Output: Soluti

Type: scalarfield

Description: Solution of the

- Types: 0 scalar, 1 scalarfield, 2 vectorfield (not yet in GUI), …

What you *can* implement – extra functionality

- Run-time parameters (not yet in GUI)

```
nlohmann::json getParameters() const override
{
  return R"([{
    "name" : "bc_east",
    "description" : "Boundary condition at the east boundary",
    "type" : "text",
    "value" : "0",
    "default" : "0",
    "uiid" : 0},{…}, {…}, {…}
    }])"_json;
}
```

What you *can* implement – extra functionality

- Update run-time parameters (triggered from the GUI)

```
nlohmann::json updateAttribute(const std::string &component,
                               const std::string &attribute,
                               const nlohmann::json &json) override {
  if (attribute == "bc_east") {
    if (!json.contains("data"))
      throw InvalidModelAttributeException();
    if (!json["data"].contains("bc_east"))
      throw InvalidModelAttributeException();

    bcFunc_[gismo::boundary::east] =
      gsFunctionExpr<T>(json["data"]["bc_east"].get<std::string>(), d);
  } else if (…) { … }
  else
    Base::updateAttribute(component, attribute, json);

  solve() // don't forget this line!!!
}
```

Let's port examples/poisson2_example.cpp

```
private:

/// @brief Base class
using Base = GismoPdeModel<d, T>;

/// @brief Type of the geometry mapping
using geometryMap_type = typename
gsExprAssembler<T>::geometryMap;

/// @brief Type of the variable
using variable_type = typename
gsExprAssembler<T>::variable;

/// @brief Type of the function space
using space_type = typename
gsExprAssembler<T>::space;

/// @brief Type of the solution
using solution_type = typename
gsExprAssembler<T>::solution;
```

```
/// @brief Multi-patch basis
gsMultiBasis<T> basis_;

/// @brief Boundary conditions
gsBoundaryConditions<T> bc_;

/// @brief Right-hand side values
gsFunctionExpr<T> rhsFunc_;

/// @brief Boundary values
std::array<gsFunctionExpr<T>, 2*d> bcFunc_;

/// @brief Expression assembler
gsExprAssembler<T> assembler_;

/// @brief Solution
gsMultiPatch<T> solution_;
```

Let's port examples/poisson2_example.cpp

```
GismoPoissonModel(const std::array<short_t, d> degrees,
                  const std::array<int64_t, d> ncoeffs,
                  const std::array<int64_t, d> npatches)
: Base(degrees, ncoeffs, npatches), basis_(Base::geo_, true),
  rhsFunc_("2*pi^2*sin(pi*x)*sin(pi*y)", d), assembler_(1, 1) {

    gsOptionList Aopt; // ... and fill it with options (not shown here)

    assembler_.setOptions(Aopt); // set assembler options

    assembler_.setIntegrationElements(basis_); // set assembler basis

    for (short_t i = 0; i < 2 * d; ++i) {
      bcFunc_[i] = gismo::give(gsFunctionExpr<T>("sin(pi*x)*sin(pi*y)", 2));
      bc_.addCondition(i + 1, gismo::condition_type::dirichlet, &bcFunc_[i]);
    }

    bc_.setGeoMap(Base::geo_); // set geometry

    solve() // don't forget this line!!!
```

The solve() function

```
void solve() {
  auto G = assembler_.getMap(Base::geo_);
  auto u = assembler_.getSpace(basis_);
  auto f = assembler_.getCoeff(rhsFunc_, G);

  u.setup(bc_, gismo::dirichlet::l2Projection, 0);

  assembler_.initSystem();
  assembler_.assemble(
    igrad(u, G) * igrad(u, G).tr() * meas(G) // matrix
    ,
    u * f * meas(G) // rhs vector
    );

  typename gismo::gsSparseSolver<T>::CGDiagonal solver;
  solver.compute(assembler_.matrix());

  gsMatrix<T> solutionVector;
  solution_type solution = assembler_.getSolution(u, solutionVector);
  solutionVector = solver.solve(assembler_.rhs());

  solution.extract(solution_); // solution as evaluatable multi-patch object
}
```

$$\int_{\Omega} \nabla u \cdot \nabla v \, dx = \int_{\Omega} uf \, dx$$

The eval() function

```
nlohmann::json eval(const std::string &component,
                    const nlohmann::json &json) const override {

  gsMatrix<T> ab = Base::geo_.patch(0).support();
  gsVector<T> a = ab.col(0);
  gsVector<T> b = ab.col(1);

  gsVector<unsigned> np(Base::geo_.parDim());
  np.setConstant(25);

  if (json.contains("data"))
    if (json["data"].contains("resolution")) {
      auto res = json["data"]["resolution"].get<std::array<int64_t, d>>();

      for (std::size_t i = 0; i < d; ++i)
        np(i) = res[i];
    }

  gsMatrix<T> pts  = gsPointGrid(a, b, np);
  gsMatrix<T> eval = solution_.patch(0).eval(pts);

  return utils::to_json(eval, true); // convert gsMatrix to JSON flattened = true
}
```

In a nutshell

- GismoYOURModel.hpp
  - Most objects from the example's main function become class members
  - Initialize members in the constructor
  - Implement getName(), getDescription(), getOutputs(), [getParameters()]
  - Implement solve() and eval() functions

- GismoYOURModel.cxx
  - Implement create() and optionally load() function

- cmake .. && make && *have fun with it*

WIP

- Deployment strategy
  - Ready-to-run docker containers for GUI and server (user & developer version)
  - Code refactoring to extract server and G+Smo modules from IgANets code
- GUI
  - More intuitive "minimalistic" user interface
  - Extension to multi-patch objects (stitching, control-point snapping, …)
  - VR/XR features (cut-planes, interactive BC and RHS control, …)
- Server & G+Smo modules
  - More applications
  - Extension to multi-patch objects

Contact me

- … if you want to try it out/contribute to it (private repos!)
- … if you have feedback on the GUI/software design
- … if you found a bug and/or have a feature request
- … if you want to connect your "solver" to the GUI

Thank you!