Efficient Pipe Interface Between the Amsterdam Modeling Suite and External Software

Tomáš Trnka[1,2], Robert Rüger[1], Ivo Durník[2] and Matti Hellström[1]

[1] Software for Chemistry & Materials B.V., Amsterdam, The Netherlands

[2] National Centre for Biomolecular Research, Faculty of Science, Masaryk University, Brno, Czechia

trnka@scm.com

About the Amsterdam Modeling Suite

AMS is a comprehensive software package for computational chemistry and material science across all scales and levels of theory:

- Common driver module for geometry optimization, advanced molecular dynamics and Monte Carlo methods, automated PES exploration and reaction discovery, ...
- Computational engines for molecular and periodic DFT, tight binding and semiempirical methods, reactive and non-reactive force fields, and machine learning potentials
- Integrated graphical user interface for job preparation, management, and analysis
- PLAMS (Python Library for Automating Molecular Simulations): an open-source package for preparing, managing, and analyzing computational chemistry calculations

Introduction

Multiscale modelling often requires coupling algorithms and potentials from different software packages:

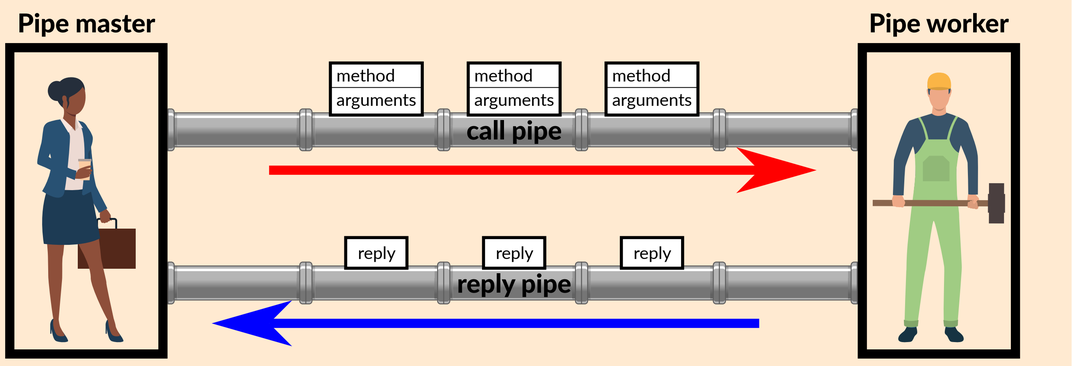
- Communication through files slow due to I/O and process startup costs
- Direct linking of all modules into a single program often infeasible for technical reasons

Solution: Universal, high-performance communication protocol as a common interface between independently developed programs.

Architecture

Two independent processes exchanging messages over a communication channel ("pipe"):

- Pipe master launches the worker and submits method calls with appropriate arguments
- Pipe worker executes the requested methods and returns any results



Images by katemangostar and pch.vector on Freepik.com

Key Features of the AMSPipe Protocol

Portability

- Works on all major platforms and across a variety of programming languages

Flexibility

- No need to implement all methods and features, only the relevant ones

Extensibility

- Methods, arguments, and reply types can be freely added without breaking compatibility

Performance

- Minimal overhead, data sent in binary, just one round trip per evaluation

Reliability

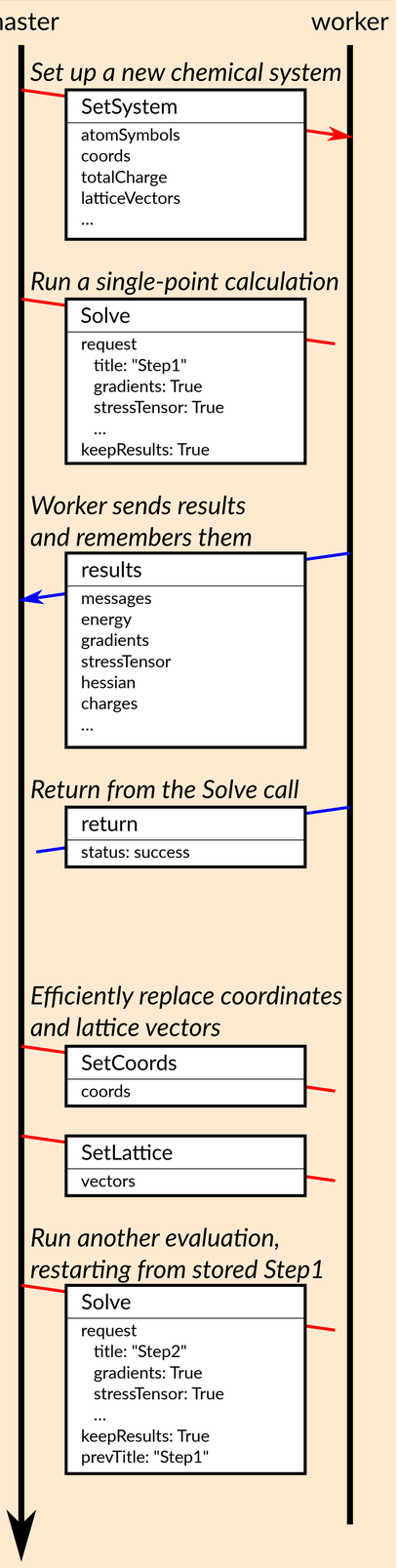
- Protocol errors are recoverable and easy to detect

Openness

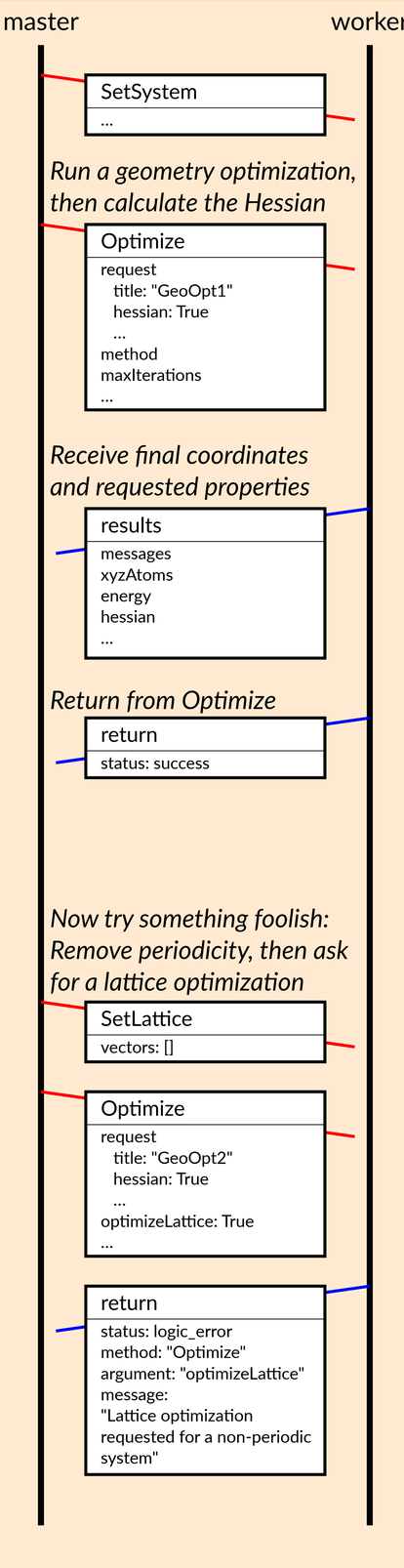
- Open-source interface modules, public protocol specification

Example Conversations

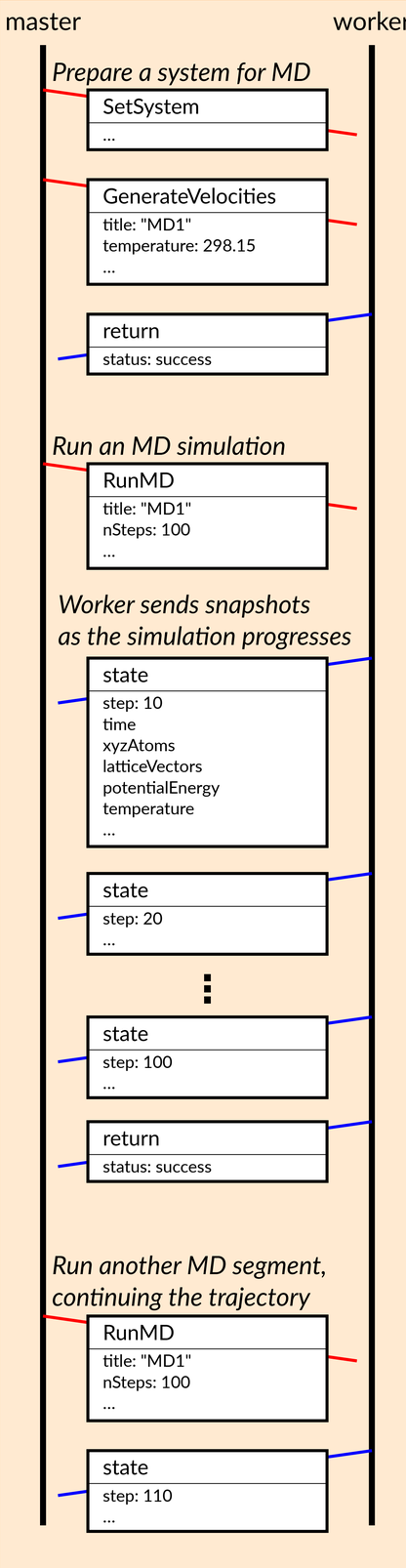
Single Points



Geometry Optimization

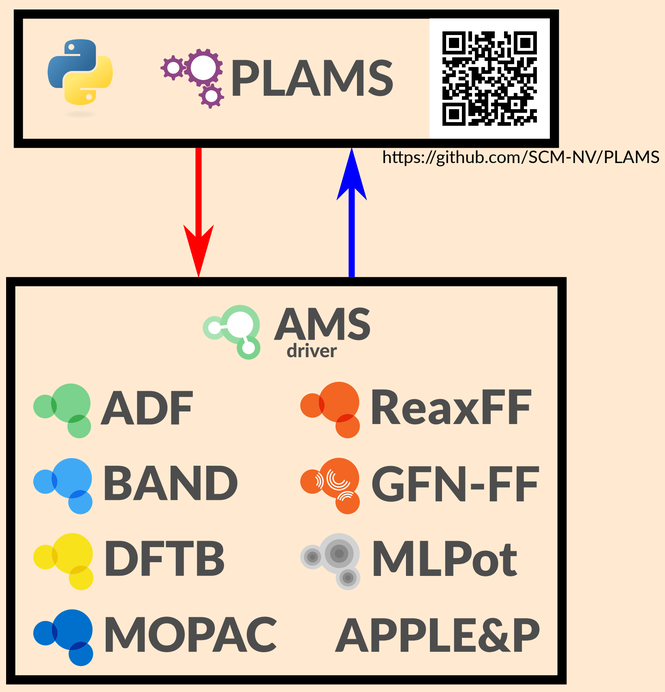


Molecular Dynamics



AMS as the Pipe Worker

The **AMSWorker** class in PLAMS enables any Python script to serve as the pipe master, offering a convenient high-level API for single points, geometry optimizations and MD simulations.



Example: GFN1-xTB energy of a ReaxFF-optimized structure

```
mol = Molecule("Water.xyz")

reax_settings = Settings()
reax_settings.input.ReaxFF.ForceField = "Water2017.ff"
dftb_settings = Settings()
dftb_settings.input.DFTB.Model = "GFN1-xTB"

with AMSWorker(reax_settings) as reax, \
     AMSWorker(dftb_settings) as dftb:
    opt_results = reax.GeometryOptimization("waterGO", mol)
    optimized = opt_results.get_main_molecule()
    dftb_results = dftb.SinglePoint("waterSP", optimized)
    print(dftb_results.get_energy())
```

AMS as the Pipe Master

- AMS driver controls the simulation, offering advanced MD and MC methods, automated PES exploration and discovery of reaction networks, multilevel parallelization of numerical (second) derivatives, properties like phonons and elastic tensors, and full GUI support
- Worker supplies an arbitrary potential

Python/ASE Workers

Any potential available as an Atomic Simulation Environment (ASE) Calculator can be readily used through the **ASEPipeWorker** class.

Used by AMS for many machine learning potentials.

Example:
Use ASE to run simulations using the EAM potential with AMS.

```
calc = ase.calculators.eam.EAM(potential="Al99.eam.alloy")
engine = scm.amspipe.ASEPipeWorker(calculator=calc)
engine.run()
```

The same defined directly in AMS input (no scripting required):

```
Engine ASE
    Type Import
    Import ase.calculators.eam.EAM
    Arguments
        potential="Al99.eam.alloy"
    End
EndEngine
```

C/C++/Fortran/... Workers

Open-source **amspipe** library offers bindings for C, C++ and Fortran, ready to be plugged into third-party SW.

Used by AMS to integrate QuantumESPRESSO, integration into LAMMPS planned.

```
AMSCallPipe   call_pipe;
AMSReplyPipe reply_pipe;

while (true) {
    auto msg = call_pipe.receive();

    if (msg.name == "Solve") {
        AMSPipe::SolveRequest request;
        bool keepResults;
        std::string prevTitle;

        call_pipe.extract_Solve(msg, request, keepResults, prevTitle);

        AMSPipe::Results results;
        std::vector<double> grads;
        results.energy = LJ_potential(coords, grads);
        if (request.gradients) {
            results.gradients = grads.data();
            results.gradients_dim[0] = 3;
            results.gradients_dim[1] = grads.size()/3;
        }

        reply_pipe.send_results(results);
        reply_pipe.send_return(AMSPipe::Status::success);
    }
}
```

Low-level Message Format

Messages are encoded using Universal Binary JSON (**UBJSON**).

UBJSON is:

- **simple**, making encoders/decoders easy to implement
- **self-describing**, unknown messages can still be parsed correctly
- **widely supported** with libraries for many languages
- somewhat **human-readable**, which simplifies debugging

Example message in JSON and UBJSON:

```
{"results": {"energy": -0.6074090031328319}}

{i\007results{i\006energyD\xbf\xe3\x6f\xe5\x01\x78\x0c\x14}}
```

i: 1-byte integer **D**: double precision float