REPORT DOCUMENTATION PAGE			*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* 17-06-2014	**2. REPORT TYPE** SBIR Phase 2 Final Report	**3. DATES COVERED** *(From - To)* November 28, 2011 - June 30, 2014

4. TITLE AND SUBTITLE Open, Cross Platform Chemistry Application Unifying Structure Manipulation, External Tools, Databases and Visualization Phase 2	**5a. CONTRACT NUMBER** W912HZ-12-C-0005
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Dr. Marcus D. Hanwell	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Kitware, Inc. 28 Corporate Drive Clifton Park, NY 12065	**8. PERFORMING ORGANIZATION REPORT NUMBER** K000695
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) US Army Engineering Research & Development Center 3909 Halls Ferry Road Vicksburg, MS 39180-6199	**10. SPONSOR/MONITOR'S ACRONYM(S)** ERDC
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRUBUTION STATEMENT A: Distribution is approved for public release; Distribution is unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Report developed under SBIR contract for topic #A10-110. The overarching goal of this project is the creation of the leading computational chemistry workbench, making the premier computational chemistry codes and databases easily accessible to chemistry practitioners. This has been accomplished by creating an open, extensible application framework that puts computational tools, data, and domain-specific knowledge at the fingertips of chemists. A data-centric approach to chemistry, storing data in a searchable database, empowers users to efficiently collaborate, innovate, and push the frontiers of research forward.

As the power of our computational resources grows, computational chemists face a growing discrepancy between our ability to run calculations/simulations and our ability to meaningfully store, search, retrieve and analyze data. As the sophistication of the computational codes grow and access to powerful computational resources becomes more commonplace, there is an increasingly steep learning curve to effectively using new computational tools and analyzing their output. Our objective is to make the lives of computational chemists easier by making these tools accessible to a wider range of chemists.

15. SUBJECT TERMS

SBIR report, computational chemistry, cheminformatics, HPC, quantum chemistry, cross platform, open source, visualization, analysis, databases, molecular dynamics, analytics, chemistry.

16. SECURITY CLASSIFICATION OF: U			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Dr. Marcus D. Hanwell
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UU	38	**19b. TELEPHONE NUMBER** *(include area code)* (518) 371-3971 ext. 520

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

Open, Cross Platform Chemistry Application

Unifying Structure Manipulation, External Tools, Databases and Visualization

Phase II SBIR Final Report for Topic A10-110
Proposal Number A2-4714

Principal Investigator
Dr. Marcus D. Hanwell

Kitware, Inc.
28 Corporate Drive
Clifton Park, NY 12065
http://www.kitware.com/
518-371-3971

30 May, 2014

Contents

1 Technical Objectives

The overarching goal of this project is the creation of the leading computational chemistry workbench, making the premier computational chemistry codes and databases easily accessible to chemistry practitioners. This has been accomplished by creating an open, extensible application framework that puts computational tools, data, and domain-specific knowledge at the fingertips of chemists. A data-centric approach to chemistry, storing data in a searchable database, empowers users to efficiently collaborate, innovate, and push the frontiers of research forward.

As the power of our computational resources grows, computational chemists face a growing discrepancy between our ability to run calculations/simulations and our ability to meaningfully store, search, retrieve and analyze data. As the sophistication of the computational codes grow and access to powerful computational resources becomes more commonplace, there is an increasingly steep learning curve to effectively using new computational tools and analyzing their output. Our objective is to make the lives of computational chemists easier by making these tools accessible to a wider range of chemists. The specific goals of the Phase II project are outlined below, and summarized in Figure 1.

- Develop an extensible, plugin-based, flexible chemistry application and library
- Develop an application for easily using HPC resources from desktop applications
- Develop a specialized desktop database application for chemical information
- Develop chemistry-specific analysis and visualization techniques
- Develop a specialized file format capable of storing large data
- Develop a library for ingesting and calculating electronic structure
- Develop a "chemistry workbench" offering state-of-the-art tools to the community

Kitware has been very successful for more than a decade by building collaborative innovation platforms that allow us to work with the best research groups in the world to leverage their research and development. This framework positions us to be able to pursue fruitful collaborations in chemistry and several other related areas.

2 Work Summary

This section summarizes the work done in the during the two-year Phase II SBIR project, with discussion of progress made in achieving the overall goals of the project, as outlined in the proposal referenced. This project involves three major areas of development (shown in Figure 1), with several open source projects supporting the work (shown in Figure 2).

The projects shown in Figure 2 summarize those being developed or extended as part of this project, with an indication of application domain and type. There are three user facing graphical applications that are aimed at being used from the desktop to do research in the broad area of computational chemistry: Avogadro 2, MongoChem, and MoleQueue. Each

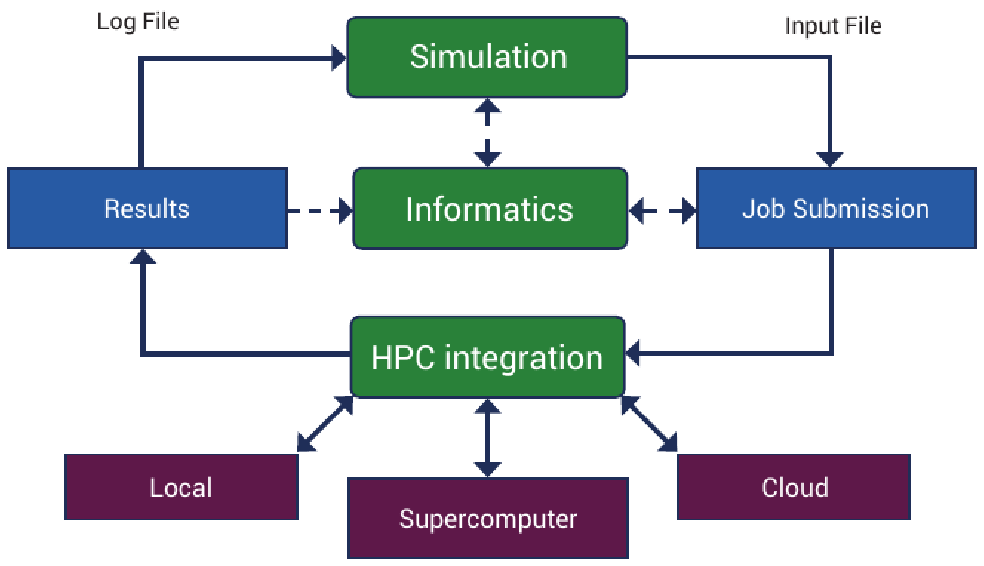


Figure 1: Open Chemistry workflow, with Open Chemistry applications filling the roles in green, and the flow of data indicated by arrows.

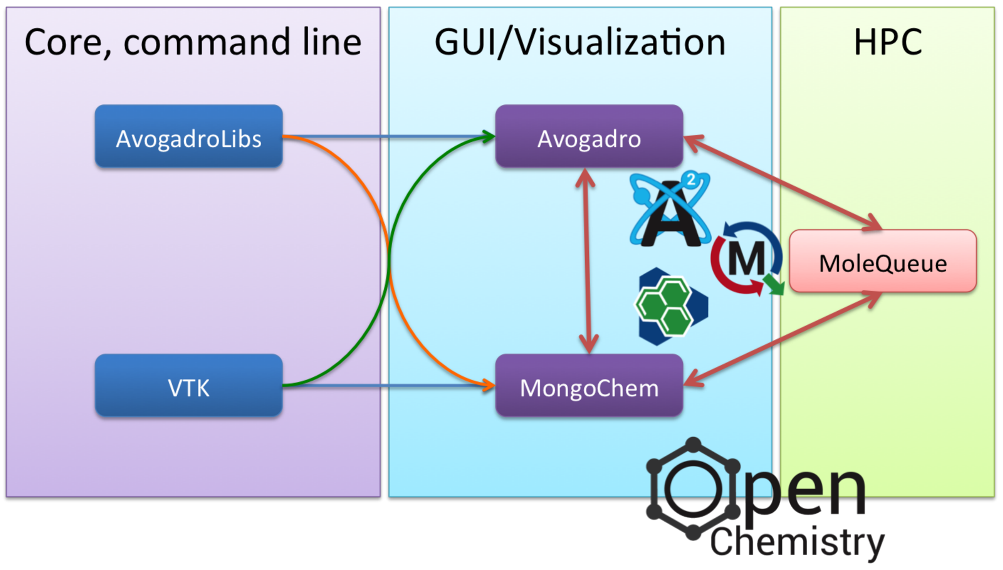


Figure 2: Open Chemistry projects grouped by basic dependency and application area.

of these applications is specialized to deal with distinct domains, but designed to be used in unison with the other applications.

These projects build upon existing libraries where possible. They are written in C++, make use of several cross-platform, open source libraries, such as Qt,[1] and CMake,[2,3] in order to build on many platforms. In addition to the more generic libraries and tools, several specialized libraries are also being developed or extended in order to support the Open Chemistry project. The VTK project[4–6] is one of the oldest C++ visualization libraries still actively developed, and is one of Kitware's core projects. It has been augmented with additional chemical data structures and visualization types that complement the existing visualization approaches in order to make it a more compelling choice for chemists. It sits between the GUI/visualization libraries and the core/command line in where it can be deployed and used—featuring both data pipeline, CPU or GPU rendering and parallel techniques—for data analysis and reduction.

The AvogadroLibs components provide the majority of the chemistry-specific functionality necessary, such as standard descriptors, file formats, force fields, data structures, algorithms, and post-processing calculations on computational chemistry output files. In addition to new functionality developed in these libraries, the Open Babel[7,8] and RDKit[9] libraries can also be used; for example, in the generation of 2D chemical structure depiction in batch mode and file format support/conversion.

2.1 Software Process and Project Dissemination

The Open Chemistry applications and libraries[10] are developed as independent projects grouped under the Open Chemistry project, with a community site at openchemistry.org. Many open source projects use a somewhat standard software process,[11] which has been adopted in a slightly modified form for the Open Chemistry projects, as shown in Figure 3.

Several key resources have been put in place for the projects:

- Community website dedicated to Open Chemistry projects
- Git source code repositories (Kitware, mirrored to Github and Gitorious)
- Online code review tool (Gerrit)
- Online software quality dashboards (CDash)
- Community wiki pages (MediaWiki)
- Bug tracking and project management tools
- Mailing lists

The community website (Figure 4) acts as an entry point to the project, and gives a brief introduction to the projects with links to specific resources. The projects use permissive, non-reciprocal BSD licenses, and distributed version control (Git) in order to enable customization of private branches and shared open branches. Git also offers the possibility of mirroring in multiple locations, with full access to the history and private mirrors possible

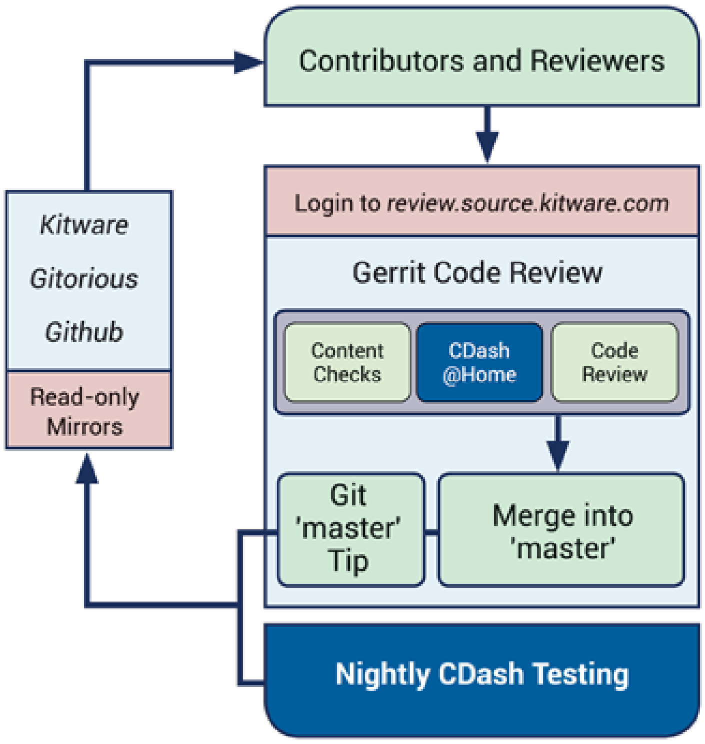
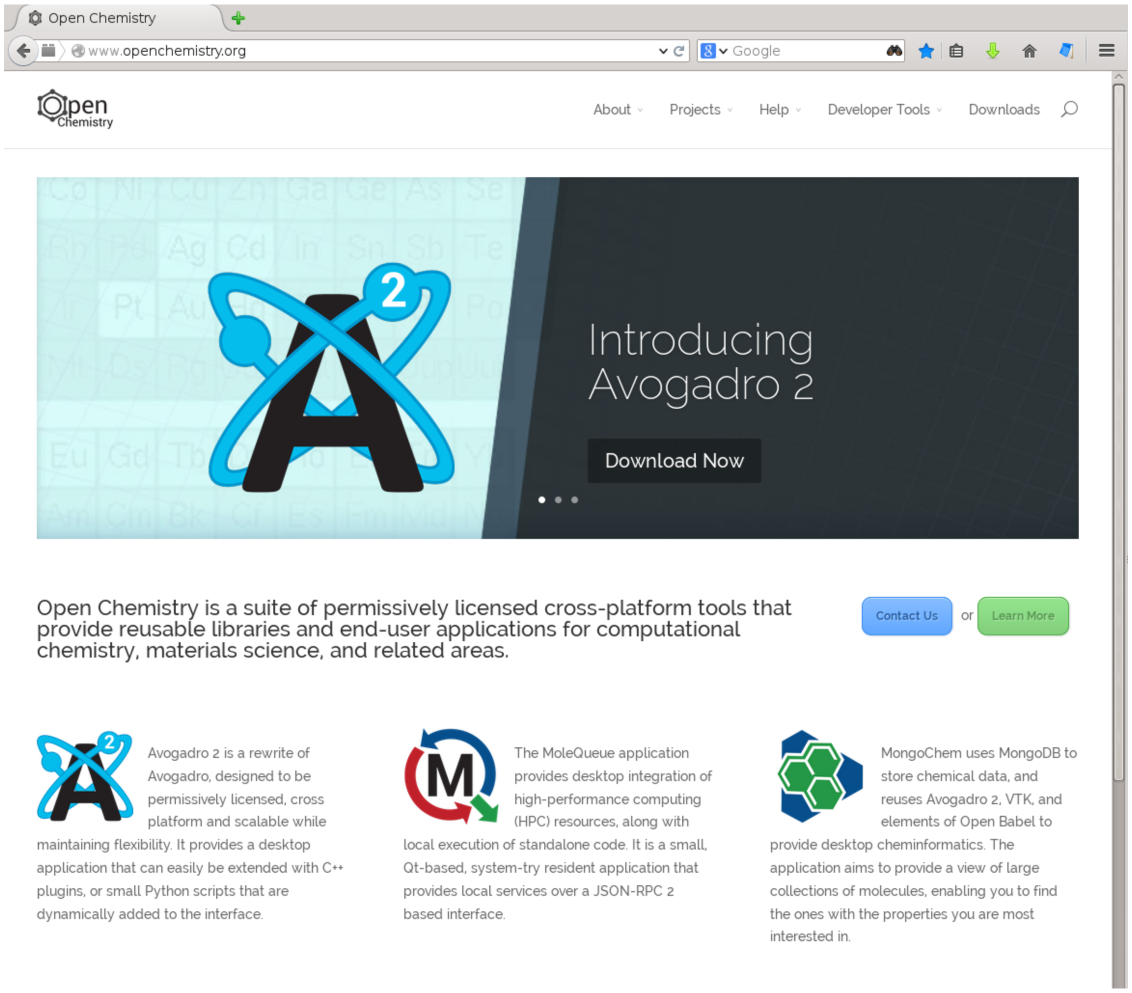


Figure 3: The software process used for Open Chemistry projects.

within organizational units. This has been successfully used with customers requiring customization that must remain private, while minimizing the maintenance/integration burden associated with traditional centralized version control systems.

The Gerrit code review system,[12] developed by Google as an open source project for the Android operating system, enables online review of code submissions from anyone while retaining control of what code is accepted into the code base. This has been combined with nightly software build testing on all three major platforms for merged code and testing of proposed changes using CDash@Home[13] (an open source project developed at Kitware to address the need for testing arbitrary branches automatically). This level of automation gives Open Chemistry projects the ability to maintain high code quality, and reviewers are free to focus on verification of code correctness while the automated systems assure that portable code that works on all major platforms. This process is summarized in Figure 3.

The projects are in independent code repositories which are then included in an Open Chemistry repository that is capable of building all of the projects, along with their major dependencies, on the mainstream platforms (Linux, Windows and Mac). The Open Chemistry repository provides an easy entry point for new developers, and using CDash with nightly testing provides binaries for Windows and Mac that are automatically generated every night. These are available both on the dashboard and on a site set up to help users find the appropriate binaries.



Open Chemistry is a suite of permissively licensed cross-platform tools that provide reusable libraries and end-user applications for computational chemistry, materials science, and related areas.

Avogadro 2 is a rewrite of Avogadro, designed to be permissively licensed, cross platform and scalable while maintaining flexibility. It provides a desktop application that can easily be extended with C++ plugins, or small Python scripts that are dynamically added to the interface.

The MoleQueue application provides desktop integration of high-performance computing (HPC) resources, along with local execution of standalone code. It is a small, Qt-based, system-try resident application that provides local services over a JSON-RPC 2 based interface.

MongoChem uses MongoDB to store chemical data, and reuses Avogadro 2, VTK, and elements of Open Babel to provide desktop cheminformatics. The application aims to provide a view of large collections of molecules, enabling you to find the ones with the properties you are most interested in.

Figure 4: The Open Chemistry homepage at openchemistry.org

2.2 Software Repositories and Statistics

Figure 2 shows the high level overview of the projects developed as part of this SBIR project, along with VTK which was extended. These artifacts can be accessed through the openchemistry.org website, shown in Figure 4 and are available for inspection by all under permissive open source licenses. The Avogadro 1.x and VTK projects remain in software repositories that were established before this project began, and both projects were enhanced as part of this work. There are also minor contributions to various projects this work depends upon.

The vast majority of the development was focused on the Open Chemistry projects which were placed in new software repositories. Some of the development statistics are summarized below for the Open Chemistry umbrella project that contains all other projects, individual statistics could also be extracted if desired. The Open Chemistry projects have:

- 2,678 commits (first recorded in March, 2011)
- 15 distinct contributors to the code
- 118,864 lines of code

It is interesting to look at the Avogadro 2 project, which is the largest part of the Open Chemistry project (nearly 65% by lines of code developed). This is due to the Avogadro Libraries repository that contains a lot of common code reused in the other applications. Taking the libraries, application, and data repositories into account:

- 1,115 commits (first recorded in October, 2011)
- 13 distinct contributors to the code
- 76,687 lines of code

These projects will be described in more detail in the following sections, the above statistics are intended only to provide some high-level numbers on the scale of the code developed, and the number of developers contributing code. In addition to these numbers the projects are leveraging code from major libraries such as VTK which has in excess of 1.47 million lines of code, from over 200 contributors with its first commit recorded in January of 1994.

2.3 Data Models and Communication Strategies

Early on in the development of the project, the Javascript Object Notation (JSON) format[14] was settled upon for simple serialization/deserialization of data, and inter-process communication. The JSON format is a simple industry standard being increasingly used in places where formats such as XML were once used. It has the distinct advantage of being a very simple format that be parsed easily, and has support in a diverse array of programming languages from compiled languages such as C, C++ and Fortran through to Java, Python, Perl, and JavaScript.

At its core, the JSON data structure consists of key/value pairs, objects, and arrays. These concepts are universal to most programming languages, enabling a great deal of freedom in language choice for data exchange. There are several C++ JSON libraries available, and two were chosen for use in the Open Chemistry project—JsonCpp which is a very small MIT licensed library using only STL, and Qt 5's JSON classes which were backported to Qt 4.8 to enable its use in Qt 4.8 and Qt 5 based projects. The Python language has native support for JSON data structures using dictionaries, and JavaScript support is strong.

The MongoDB project[15] was chosen as a scalable NoSQL data store for the cheminformatics components of this work. The MongoDB project uses a binary form of JSON called BSON,[16] which follows many of the same principles as JSON, but stores data in raw binary and has optimized the data structures to support fast reading and writing of documents. This means that moving data from the backend data store to applications and over inter-process communication channels is a simple process. Several libraries are also available with native BSON support, such as C, C++, and Python. BSON has the distinct advantage of IO speed and the ability to store raw binary data, such as PNG images, binary file fragments etc. with binary data length encoded in the standard representation and support for most basic types.

The use of JSON and BSON in these projects also prompted the development of a JSON/BSON data model to represent chemical structures. This was developed to mirror many of the structures already developed for the Chemical Markup Language, and translation between the two formats should be lossless. They are both extensible formats building on widely-accepted industry standard data exchange formats. Neither is especially suited to very large data, but can be coupled with a separate binary data store to give semantically rich data documents that point to larger blobs of binary data when appropriate. An example of a small structure in Chemical JSON:

```
{
  "chemical json": 0,
  "name": "Ethane",
  "inchi": "1/C2H6/c1-2/h1-2H3",
  "formula": "C 2 H 6",
  "atoms": {
    "elements": {
      "number": [  1,   6,   1,   1,   6,   1,   1,   1 ]
    },
    "coords": {
      "3d": [  1.185080, -0.003838,  0.987524,
               0.751621, -0.022441, -0.020839,
               1.166929,  0.833015, -0.569312,
               1.115519, -0.932892, -0.514525,
              -0.751587,  0.022496,  0.020891,
              -1.166882, -0.833372,  0.568699,
              -1.115691,  0.932608,  0.515082,
              -1.184988,  0.004424, -0.987522 ]
    }
  },
  "bonds": {
    "connections": {
      "index": [ 0, 1,
                 1, 2,
                 1, 3,
                 1, 4,
                 4, 5,
                 4, 6,
                 4, 7 ]
    },
    "order": [ 1, 1, 1, 1, 1, 1, 1 ]
  },
  "properties": {
    "molecular mass": 30.0690,
    "melting point": -172,
    "boiling point": -88
  }
}
```

The key names map well to the XML nodes in CML documents,[17] and this structure can easily be stored directly in MongoDB as a document (or object within a document) or passed between processes as JSON. Readers can check for the existence of known keys, and JSON documents can be built up by various subroutines using a simple in-memory model. This format maps well to BSON documents where each key/value has a type and length field before the actual data, using arrays of floats, doubles, integers, etc. maximizes storage efficiency, binary read speed and ability to skip 3d coordinates efficiently when properties are the key of interest for example. The equivalent CML is shown below to aid comparison.

```
<?xml version="1.0" encoding="UTF-8"?>
<molecule xmlns="http://www.xml-cml.org/schema"
          xmlns:cml="http://www.xml-cml.org/dict/cml"
          xmlns:units="http://www.xml-cml.org/units/units"
          xmlns:xsd="http://www.w3c.org/2001/XMLSchema"
          xmlns:iupac="http://www.iupac.org"
          id="CS_ethane">
  <formula concise=" C 2 H 6 "/>
  <identifier convention="iupac:inchi" value="1/C2H6/c1-2/h1-2H3"/>
  <name convention="IUPAC">Ethane</name>
  <atomArray>
    <atom id="a1" elementType="H" x3="1.185080" y3="-0.003838" z3="
       0.987524"/>
    <atom id="a2" elementType="C" x3="0.751621" y3="-0.022441" z3="
       -0.020839"/>
    <atom id="a3" elementType="H" x3="1.166929" y3="0.833015" z3="
       -0.569312"/>
    <atom id="a4" elementType="H" x3="1.115519" y3="-0.932892" z3="
       -0.514525"/>
    <atom id="a5" elementType="C" x3="-0.751587" y3="0.022496" z3="
       0.020891"/>
    <atom id="a6" elementType="H" x3="-1.166882" y3="-0.833372" z3="
       0.568699"/>
    <atom id="a7" elementType="H" x3="-1.115691" y3="0.932608" z3="
       0.515082"/>
    <atom id="a8" elementType="H" x3="-1.184988" y3="0.004424" z3="
       -0.987522"/>
  </atomArray>
  <bondArray>
    <bond atomRefs2="a1 a2" order="1"/>
    <bond atomRefs2="a2 a3" order="1"/>
    <bond atomRefs2="a2 a4" order="1"/>
    <bond atomRefs2="a2 a5" order="1"/>
    <bond atomRefs2="a5 a6" order="1"/>
    <bond atomRefs2="a5 a7" order="1"/>
    <bond atomRefs2="a5 a8" order="1"/>
  </bondArray>
  <propertyList>
    <property dictRef="cml:molwt" title="Molecular weight">
      <scalar dataType="xsd:double" dictRef="cml:molwt" units="units:g
         ">30.0690</scalar>
    </property>
    <property dictRef="cml:monoisotopicwt" title="Monoisotopic weight"
       >
      <scalar dataType="xsd:double" dictRef="cml:monoisotopicwt" units
         ="units:g">30.0469502</scalar>
    </property>
    <property dictRef="cml:mp" title="Melting point">
      <scalar dataType="xsd:double" errorValue="1.0" dictRef="cml:mp"
         units="units:celsius">-172</scalar>
    </property>
    <property dictRef="cml:bp" title="Boiling point">
      <scalar dataType="xsd:double" errorValue="1.0" dictRef="cml:bp"
         units="units:celsius">-88</scalar>
    </property>
  </propertyList>
</molecule>
```

2.4 Avogadro 1.x

The Avogadro 1.x project[18] was developed as a library and desktop application since its inception. A paper was published in 2012 discussing the work that went into the project leading up to the 1.0 release series,[19] providing a summary of the development effort. As of this writing the paper has received 184 citations according to Google Scholar, and is the third most viewed paper of all time in the Journal of cheminformatics with Figure 5 showing the graphical abstract. After the paper was published the Avogadro 1.1.0 release was made, which incorporated some of the early improvements developed during this SBIR project (largely in Phase I, and early in Phase II). Development in Phase II of the project was split between improvements to and stabilization of features in Avogadro 1.1, with the majority of effort redirected to rewriting the core data structures and algorithms for Avogadro 2, which moves from a GPLv2 only license to a more permissive 3-clause BSD license that allows for much wider use in all sections of government, academia, industry, and education.

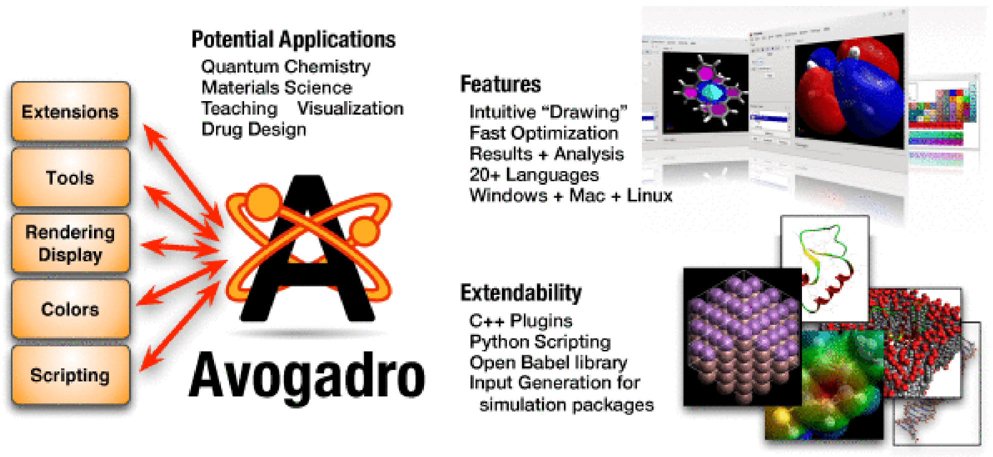


Figure 5: The graphical abstract for the Avogadro paper.

The Avogadro application is a user-facing application capable of loading, editing and saving chemical structures and loading/analyzing output from many popular computational chemistry codes. It is developed as an open source community project, with input from across the industry in both research and education. In the latest release of the Avogadro project (1.1.0) significant new features were added, such as support for directly reading GAMESS-US log files among other new codes, automated calculation of all molecular orbital intensities in the background to enable the rapid comparison of orbitals shapes/size, and improved support for vibrational mode animations. A new crystallography extension was added, providing significantly improved support for periodic structures. A crystal library was added to the distribution, along with new builders such as a nanotube builder and chirality inversion.

2.5 Avogadro 2 Libraries

The Avogadro 1.1 branch continues to be developed, but development efforts moved to the Avogadro 2 rewrite (along with porting code). All major contributors agreed to relicense their contributions under the 3-clause BSD license, with a new set of libraries developed to serve the next generation of chemical manipulation and visualization tools. In order to serve everything from desktop applications to HPC client-server deployments, the libraries have been developed using a much more modular pattern. In Avogadro 1 there was a single Avogadro library, and an application that linked to it. Avogadro 2 features a number of libraries for specific application areas. This is needed as heavy computation needs to take place on HPC systems with no graphical environment, whereas the desktop applications needs a range of custom rendering and graphical widgets in order to be user friendly and easy-to-use.

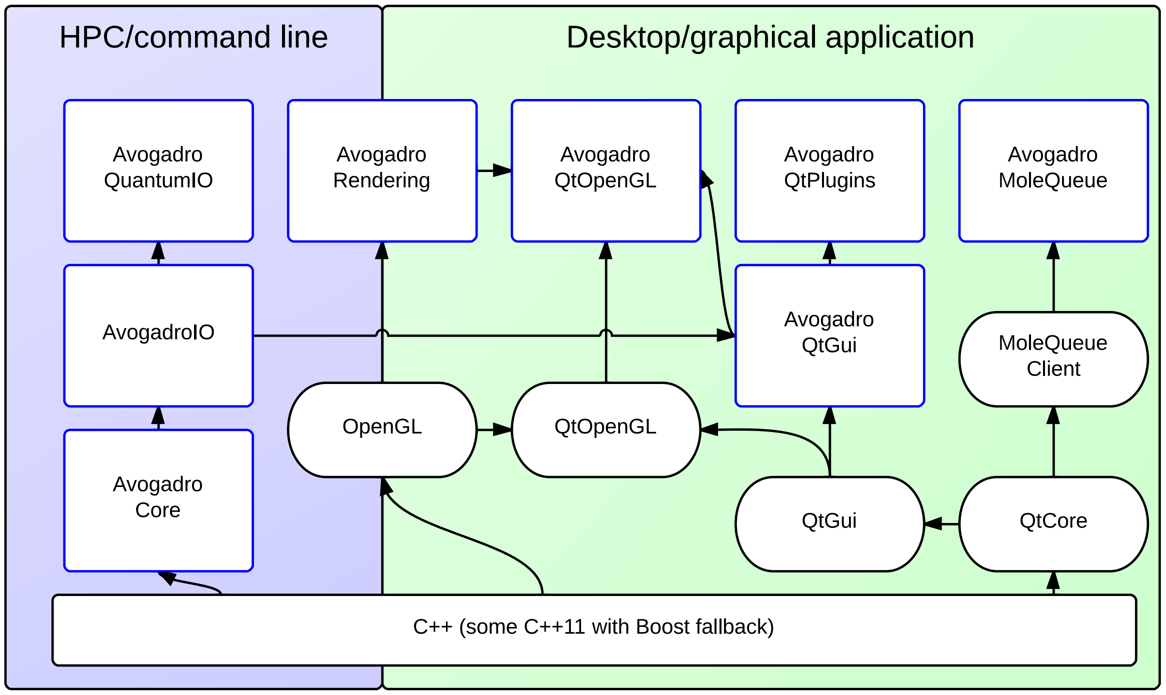


Figure 6: The organization of the Avogadro 2 libraries (blue outlines), and some of their dependencies.

Avogadro 2 is a complete rewrite of the libraries and applications from the ground up. All of the core data structures, APIs, rendering algorithms, file handling, plugin architecture and interaction were rethought and written for both scalability and streaming. This is most apparent when looking at large systems in Avogadro 1.x and Avogadro 2—systems exceeding a few thousand atoms in size were difficult to load and/or interact with previously, becoming very apparent above ten thousand atom systems. Avogadro 2 has been demonstrated loading and interacting with systems containing in excess of 2.8 million atoms, with frame rates remaining interactive and load times reasonable. These represent some of the largest systems most groups are interested in simulating using molecular dynamics at the current time, and it is clear that it would be possible to look at even larger systems.

2.5.1 Core and IO Libraries

The load and interaction improvements have been achieved by building a core library that has very few dependencies, avoiding any graphical operations. Data structures, core algorithms, and common definitions are implemented here. A small, focused input/output library extends the core library and adds basic file IO, with new dependencies on small JSON and XML parsing libraries, and Boost in order to efficiently implement core file format support. The HDF5 library is needed by the IO library; most other file formats will be translated to these core formats using tools such as Open Babel, Chemkit, and RDKit. The file IO layer is extensible, and manages registered format handlers implementing some project specific formats such as Chemical JSON described earlier, as well as common chemical file formats such as CML, XYZ, MDL, and the GROMACS .gro format. A snippet from the XYZ format shows how simple the read function is, where the format developer implements a simple method taking the input stream and a molecule object as parameters, and populates the molecule object.

```
bool XyzFormat::read(std::istream &inStream, Core::Molecule &mol)
{
  size_t numAtoms = 0;
  if (!(inStream >> numAtoms)) {
    appendError("Error parsing number of atoms.");
    return false;
  }

  std::string buffer;
  std::getline(inStream, buffer); // Finish the first line
  std::getline(inStream, buffer);
  if (!buffer.empty())
    mol.setData("name", trimmed(buffer));

  // Parse atoms
  unsigned char atomicNum;
  Vector3 pos;
  for (size_t i = 0; i < numAtoms; ++i) {
    if (inStream >> buffer &&
        inStream >> pos.x() &&
        inStream >> pos.y() &&
        inStream >> pos.z()) {
      if (!buffer.empty()) {
        if (isalpha(buffer[0])) {
          atomicNum = Elements::atomicNumberFromSymbol(buffer);
        }
        else {
          short int atomicNumInt = 0;
          std::istringstream(buffer) >> atomicNumInt;
          atomicNum = static_cast<unsigned char>(atomicNumInt);
        }
        Atom newAtom = mol.addAtom(atomicNum);
        newAtom.setPosition3d(pos);
        continue;
      }
    }
    break;
  }

  // Check that all atoms were handled.
  if (mol.atomCount() != numAtoms) {
    std::ostringstream errorStream;
    errorStream << "Error parsing atom at index " << mol.atomCount()
                << " (line " << 3 + mol.atomCount() << ").";
    appendError(errorStream.str());
    return false;
  }

  return true;
}
```

A fast and lightweight CML reader and writer has been developed using the PugiXML library for XML parsing, and the HDF5 library for storing large amounts of data in a binary format. Significant improvements in load time and memory utilization have been achieved over previous implementations, along with simpler code that can be more easily extended in the future as more features are required. The CML and HDF5 reader/writer has been developed as part of this project, and discussed with experts in the field as an approach to creating standards-compliant and scalable formats for use in chemistry. Converters from other formats are also being developed using two approaches: the extension of Open Babel which already supports a vast array of formats, and the development of JUMBO converters in collaboration with others in the field directly to CML.

2.5.2 Molecule Classes

Previously the molecule class of Avogadro was deeply entwined with that of Open Babel, and one class was used for everything. In Avogadro 2 after a great deal of thought three molecule classes were created. The core Avogadro library houses the first, which implements a Core::Molecule class. A flyweight proxy pattern was employed for the molecule's atoms and bonds, in stark contrast to previously used approaches. Atoms and bonds are temporary objects, containing only a pointer to their parent molecule and their index—along with all expected API. The molecule has a series of data arrays that contain the atom's atomic number, position, etc. Properties that are not used remain as zero-length arrays in the molecule, and are only allocated when values are set.

The molecules leverage a copy-on-write pattern for heavy data storage, whereby a copy of a molecule will create a copy of all arrays in the old molecule in the new one. The arrays increment the reference count of the inner data, but do not make a copy of the memory unless a non-const method is called. In this way very cheap copies of large molecules can be created, and temporary molecules are able to effectively pass along their underlying data without creating multiple, redundant, in-memory copies of heavy data. This also makes file input/output much more efficient as iterating through all stored position coordinates is a simple traversal of a linear array, allocating these buffers for a known input can involve just one initial memory allocation reducing memory fragmentation. The code listing below demonstrates how simple it is to create a molecule, and work with the proxy atom/bond objects in the molecule classes.

```
Avogadro::Core::Molecule mol;
Avogadro::Core::Atom o1 = mol.addAtom(8);
Avogadro::Core::Atom h2 = mol.addAtom(1);
Avogadro::Core::Atom h3 = mol.addAtom(1);

o1.setPosition3d(Vector3(0, 0, 0));
h2.setPosition3d(Vector3(0.6, -0.5, 0));
h3.setPosition3d(Vector3(-0.6, -0.5, 0));

Avogadro::Core::Bond b1 = mol.addBond(o1, h2, 1);
Avogadro::Core::Bond b2 = mol.addBond(o1, h3, 1);
```

The QtGui::Molecule class inherits from Core::Molecule and Qt's QObject so that it can participate in the Qt framework. It adds signals and slots, as well as the simple parenting offered by the Qt framework for object lifetime management. This is the object used primarily by the Avogadro 2 application; it is passed to the file formats which take the molecule to be populated as an input (although they are all implemented in terms of the Core::Molecule class.

The third is QtGui::RWMolecule which inherits from just QObject, but shares a common API with the other two classes. The QtGui::Molecule class and QtGui::RWMolecule classes offer fast conversion from one to the other, and the "editable" molecule is highly specialized for molecular editing. It only supports atoms, bonds, and limited metadata, leaving more advanced objects such as quantum data, cubes, meshes etc. to the other classes. It builds and maintains an internal undo/redo stack, removing the burden from the mouse interaction tools to implement undo/redo operations. It uses the same array classes that offer copy-on-write functionality, meaning that conversion to QtGui::Molecule copies very little data unless further changes are made to the editable molecule.

The molecule classes are also supported by support classes that provide atomic data, such as atomic radii, default colors, etc. These functions are significantly faster due to their use of linear arrays indexed by atomic number, further increasing efficiency when working with large structures.

2.5.3 Periodic Structures

Support for periodic systems was implemented, with options to display and edit the unit cell, as well as perform numerous operations on the cell. These concepts were added as optional properties on molecules, and several of the native file formats include full support for expressing periodic structure. Figure 14 shows a large structure from a GROMACS simulation with the unit cell for the periodic boundaries displayed.

2.5.4 Rendering and Graphical Libraries

The rendering data structures, support classes, and code reside in a rendering library that depends upon OpenGL and GLEW, but remains independent of the graphical user interface toolkit employed. This opens up the possibility of deploying the rendering code in a wider variety of environments, but requires integration with Qt in order to open up windows, handle user interaction, and perform other common tasks. This is implemented in a Qt OpenGL library that provides customized OpenGL render windows and related functionality. Finally, the desktop widgets necessary for user interaction without OpenGL is in a Qt GUI library, to enable reuse in non-OpenGL applications. Plugin location, loading, and lifetime management are also implemented in the Qt GUI library.

Significant advances in the rendering model have focused on leveraging OpenGL 2.1 in order to maximize performance with good hardware support. Major advantages include vertex buffer objects, vertex and fragment shader programs, and optimized memory layouts for rendering. One of the largest bottlenecks in the rendering pipeline in chemical structures is sphere rendering, which has been significantly mitigated through the use of impostor sphere rendering. This approach uses a point sprite or single quad to represent a sphere, the vertex program applies a billboarding transform to ensure the quad faces the camera and has the correct dimensions in eye space. The fragment shader then applies lighting equations and an implicit function to update the depth buffer with the correct values to interact with standard rendering techniques. This leads to a highly optimized rendering scene where only four points per sphere need to be transformed; and sphere ray-tracing equations can be applied on a per pixel basis, offering not only much improved rendering speed, but pixel perfect sphere boundaries due to the use of an implicit sphere rather than some approximate triangulation method as is traditionally used. The results can be seen in Figure 7.

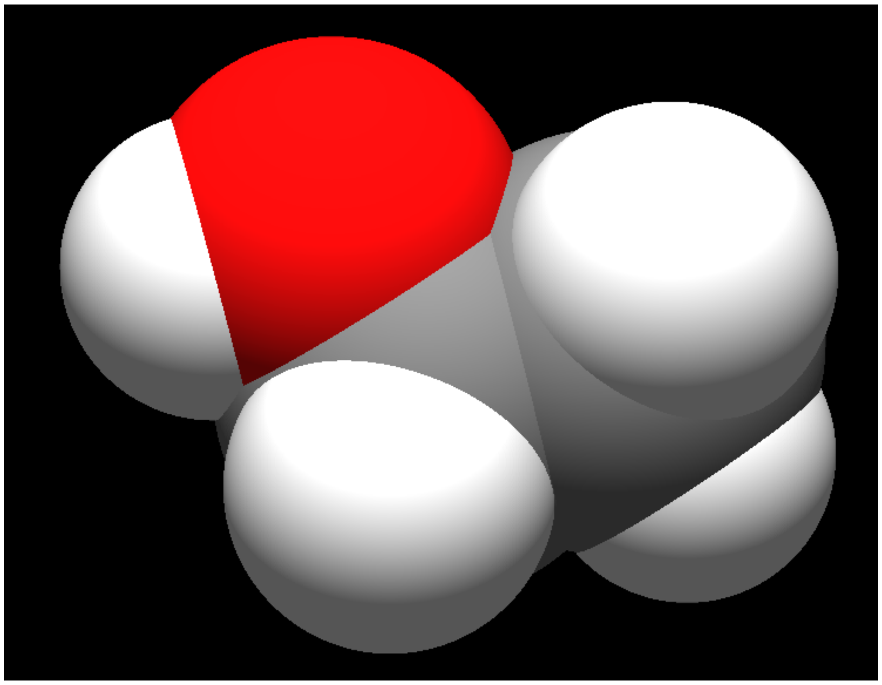


Figure 7: Van der Waals rendering using impostor sphere rendering.

Ideally a similar approach would be used for cylinders, but the gains are lower (cylinders are more complex to model in this way and require fewer triangles than spheres). Figure 8 shows a typical ball and stick representation, and the seamless joins achieved between the ray-traced spheres and the triangulated cylinders. Through the use of optimized data structures and vastly improved rendering techniques structures containing hundreds of thousands of atoms can now be rendered interactively on commodity laptops when in Avogadro 1.x thousands of atoms would already be displaying performance degradation.

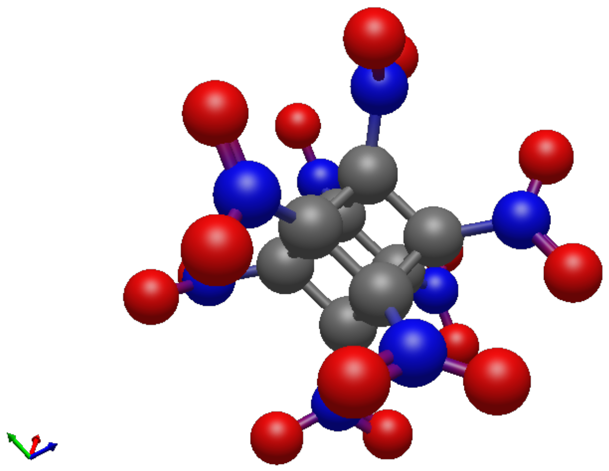


Figure 8: Ball and stick rendering using impostor spheres and triangulated cylinders, saved directly to PNG from the application.

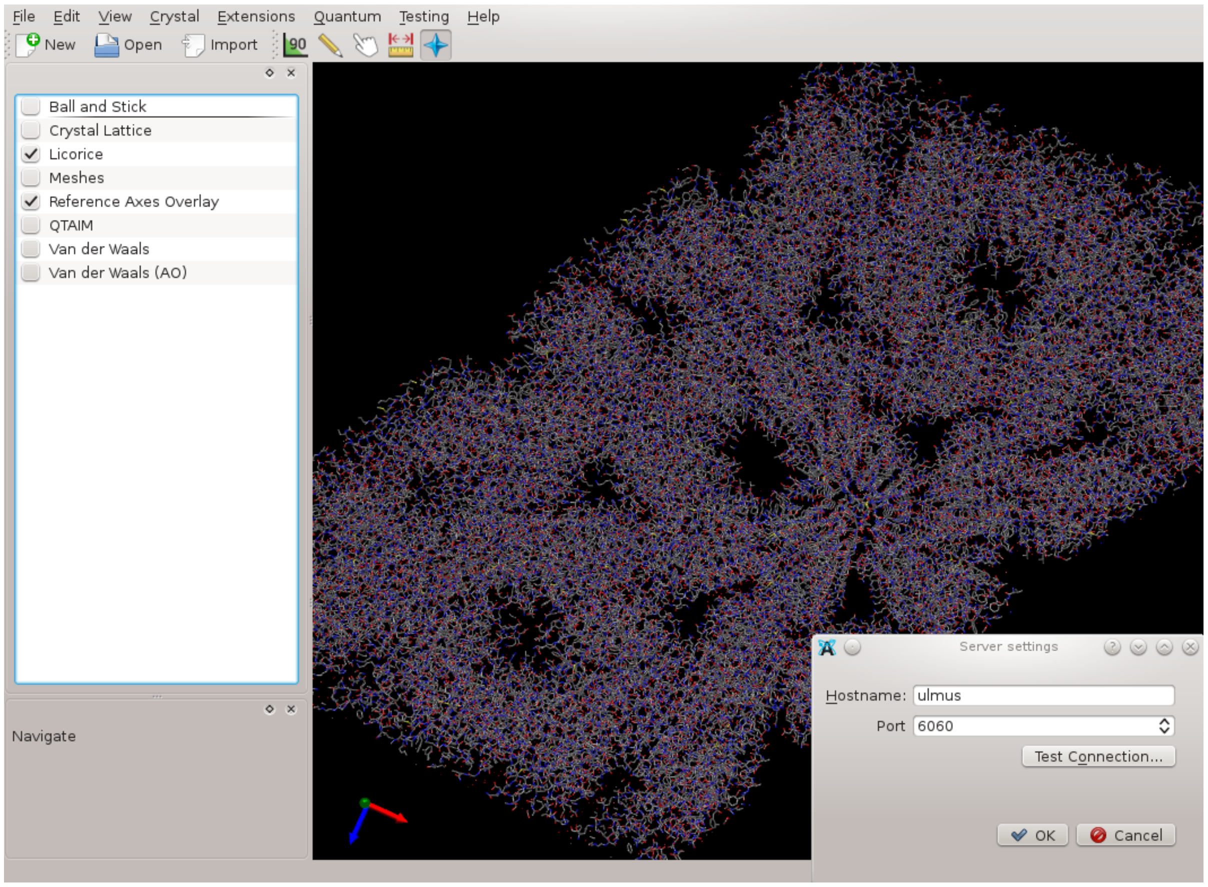
The improvements in low level rendering capabilities using OpenGL 2.1, and modern shader language approaches are further augmented through the use of a custom scene graph implementation. The scene graph is becoming increasingly accepted as the best pattern to take maximum advantage of modern graphics cards, and is commonly used in many rendering engines from the latest blockbuster video games through to rendering engines for movies and cartoons to real-time ray tracing engines. This adds a significant amount of API and infrastructure to the rendering code in Avogadro 2, but pays off with a simple user-facing API coupled with extremely efficient batching of rendering.

In a typical scene any given view might use spheres, cylinders, triangle meshes, and text. Using the traditional immediate mode techniques these graphical primitives would be rendered in the order they were encountered, but with a scene graph multiple passes are used. The first pass involved going from the molecular model to the graphical model, which is where view plugins transform atoms, bonds, surfaces, etc. into spheres, cylinders, triangle meshes, etc. This can now be pushed off into background threads as they are simply building in-memory representations of the molecule ready for the rendering pass. On the first rendering pass the scene graph drawable items translate the graphical primitives into vertex buffer objects (VBOs) that are uploaded to fast GPU memory, along with the appropriate uniforms and shader programs necessary to render.

Once the geometry has been uploaded to GPU-resident memory the actual render calls act in large batched operations—draw all spheres, draw all cylinders, draw all meshes, etc. These calls apply the shader code to the geometry in memory, and when the camera is changed to look at the structure from a different angle or zoom level the camera matrix is updated but all other state remains unaffected, meaning that virtually no data needs to be uploaded to the graphics card for the next frame. This coupled with the orders of magnitude decrease in vertex counts for spheres has enabled rendering to go from thousands of spheres in Avogadro 1.x to millions of spheres in Avogadro 2. The significantly improved core molecule data structure (with all 3D coordinates being in contiguous memory for example) also enables for significantly improved initial render times.

2.5.5 Client-Server and Interprocess Communication

In addition to lower level changes to data structures, rendering and plugins, a scalable, client-server oriented architecture has been developed, yielding fast serialization/deserialization of data and simple migration of objects from one process to another (whether local or remote). This architecture is exposed as a generic library that leverages Google's protobuf project for fast, binary communication, with specialized helper classes in Avogadro 2 libraries to make transfers of common data structures simpler. There is a runtime loaded plugin that exposes this in the application, and facilities for remote file browsing with all communication happening over a standard TCP/IP connection. The first screen capture of this is shown in Figure 9.



- [ ] Ball and Stick
- [ ] Crystal Lattice
- [x] Licorice
- [ ] Meshes
- [x] Reference Axes Overlay
- [ ] QTAIM
- [ ] Van der Waals
- [ ] Van der Waals (AO)

Hostname: ulmus
Port 6060

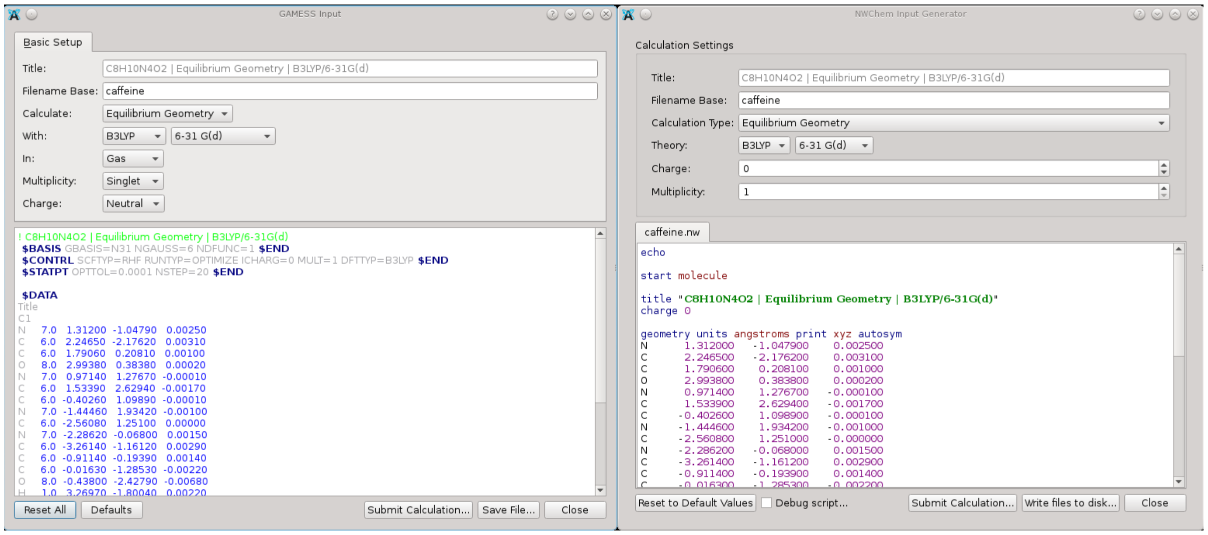
Figure 9: The Avogadro 2 application displaying a molecule loaded on a remote system using the client-server functionality.

Integration with MoleQueue and MongoChem offers seamless communication of molecular data between components, opening selected results from MongoChem in Avogadro 2, or new calculations performed by MoleQueue automatically in Avogadro 2 once they are finished. The original code was developed in the MoleQueue project (described later), and generalized so that all three applications could easily set up local socket connections and listen for JSON-RPC 2.0 calls. The code implementing the calls is simple, and can be extended with minimal effort to support new interactions as desired.

2.5.6 Input Generators in Separate Processes

In addition to the previous approach taken to adding new input generators using C++, development of a new methodology has also been developed. Instead of writing a plugin in C++ or Python using the wrapped API of Avogadro, calling a script directly from the Avogadro application in a separate process is now possible.

This suffers from a higher startup cost, but benefits greatly from the level of simplicity in designing, adding and editing input generators. All molecular geometry and calculation settings can be passed in using a JSON input to the Python (or any other language) process, and the generated output can be passed back to the application using the standard output of the process. Most languages have support for JSON, and can parse it very efficiently. Calling an independent process removes any issues around consistent linking of the plugin, licensing issues and complexity of learning a new API. This approach can be used in combination with the previous approach, with some input file generators being C++ plugins, and others being implemented in these independent scripts. Should the plugin crash or be unreliable it will not crash the main application process, and the user can be informed of the issue. Examples of C++ and Python input generators are shown in Figure 10—both feature syntax highlighting.



Title: C8H10N4O2 | Equilibrium Geometry | B3LYP/6-31G(d)
Filename Base: caffeine
Calculate: Equilibrium Geometry
With: B3LYP 6-31 G(d)
In: Gas
Multiplicity: Singlet
Charge: Neutral

```
! C8H10N4O2 | Equilibrium Geometry | B3LYP/6-31G(d)
 $BASIS GBASIS=N31 NGAUSS=6 NDFUNC=1 $END
 $CONTRL SCFTYP=RHF RUNTYP=OPTIMIZE ICHARG=0 MULT=1 DFTTYP=B3LYP $END
 $STATPT OPTTOL=0.0001 NSTEP=20 $END

 $DATA
Title
C1
N   7.0  1.31200 -1.04790  0.00250
C   6.0  2.24650 -2.17620  0.00310
C   6.0  1.79060  0.20810  0.00100
O   8.0  2.99380  0.38380  0.00020
N   7.0  0.97140  1.27670 -0.00010
C   6.0  1.53390  2.62940 -0.00170
C   6.0 -0.40260  1.09890 -0.00010
N   7.0 -1.44460  1.93420 -0.00100
C   6.0 -2.56080  1.25100  0.00000
N   7.0 -2.28620 -0.06800  0.00150
C   6.0 -3.26140 -1.16120  0.00290
C   6.0 -0.91140 -0.19390  0.00140
C   6.0 -0.01630 -1.28530 -0.00220
O   8.0 -0.43800 -2.42790 -0.00680
H   1.0  3.26970 -1.80040  0.00220
```

Title: C8H10N4O2 | Equilibrium Geometry | B3LYP/6-31G(d)
Filename Base: caffeine
Calculation Type: Equilibrium Geometry
Theory: B3LYP 6-31 G(d)
Charge: 0
Multiplicity: 1

```
echo

start molecule

title "C8H10N4O2 | Equilibrium Geometry | B3LYP/6-31G(d)"
charge 0

geometry units angstroms print xyz autosym
N      1.312000   -1.047900    0.002500
C      2.246500   -2.176200    0.003100
C      1.790600    0.208100    0.001000
O      2.993800    0.383800    0.000200
N      0.971400    1.276700   -0.000100
C      1.533900    2.629400   -0.001700
C     -0.402600    1.098900   -0.000100
N     -1.444600    1.934200   -0.001000
C     -2.560800    1.251000   -0.000000
N     -2.286200   -0.068000    0.001500
C     -3.261400   -1.161200    0.002900
C     -0.911400   -0.193900    0.001400
C     -0.016300   -1.285300   -0.002200
```

Figure 10: The GAMESS input generator (left) implemented in C++, and NWChem (right) implemented in Python.

This also opens up the possibility of much simpler packaging, sharing and independent releases of input generator plugins. They can be downloaded and placed in an appropriate location. Avogadro 2 takes care of calling them using several entry points asking for the menu entry to be added to the application, supported options and desired molecular geometry format. The plugins can also return syntax highlighting rules, which are then loaded by the input generator and shown in real time when the user hand edits the input file. Concerns over how well this might scale to larger structures have also been mitigated by providing a geometry specification syntax, where the plugin can specify how the molecular geometry should be passed and offering a known keyword for later replacement. The framework also supports the generation of multiple input files.

2.5.7 File Format Extensions in a Separate Process

Early on in Avogadro 2's development there was a strong desire to maintain support for the large number of file formats supported by Open Babel without linking to the Open Babel library due to licensing and program stability concerns. This led to the design of a meta-format plugin that queried the obabel command line executable for all supported formats, added them to the Avogadro 2 formats, and integrated them into the applications IO routines. This was achieved, and found to work well using obabel to convert to CML, XYZ, or MDL as appropriate making a call in a separate process to obabel for each file to be opened or saved.

This approach was then extended, following a similar pattern to the input generators in a separate process described above. There are a set of known entry points that must be implemented, informing the application if the format can read, write, or both, along with the requested format for writing or the format the application should expect for reading. The files are then passed to the plugin as requested, which is expected to perform the translation to/from the format it implements. This again enables the simple extension of the application using Python scripts (or any other language), and new formats can be seamlessly added to the application. From the end user's perspective these plugins are indistinguishable from native file formats and/or input generators, though it should be noted they suffer from the inherent overhead of starting a distinct process for each call, and use the standard input/output streams to avoid complications with temporary files.

2.5.8 OpenQube—Moved into Avogadro Libraries

The OpenQube project began as part of an Avogadro plugin, and was later split out in order to make it more useful in other applications. It originally contained a minimal molecule data structure, which has since been ported to reuse the structure implemented in Avogadro::Core and moved to a library in the Avogadro 2 libraries repository—Avogadro::Quantum. Support for a variety of output file formats, such as GAMESS, GAMESS-UK, MOPAC, and Molden have been added. Experimental support for CMLComp is also being developed, with a collaboration between NWChem, Open Chemistry, and community members exploring augmenting NWChem and Avogadro 2 with CMLComp support, and developing new converters to go from log file formats to the CMLComp format. The CMLComp convention is being developed in a larger collaboration, with XML dictionaries being developed to extend CML for use in computational chemistry.

Further generalization of OpenQube also took place, in order to develop a more widely-applicable and efficient data structure where basis sets can be shared between multiple atoms, support for UHF as well as RHF and higher order Gaussian type orbital functions. The OpenQube code had a hard dependency on Qt for the QtConcurrent parallelization framework, this was removed with a simple serial implementation in the core library and the parallel QtConcurrent based approach moved to an Avogadro 2 plugin. This adds the possibility of client-server molecular orbital and electron density calculations, which are well suited to this approach due to the highly CPU bound nature of the calculations.

2.5.9 Mouse Interaction Tools

Due to the creation of multiple molecule classes the complexity of the tools plugins increased a little, but this was offset by the centralization of the undo/redo management. Several tools were ported from Avogadro 1.x, although these largely became rewrites due to significant changes to the core APIs, and simplifications that became more obvious when reexamining previous approaches.

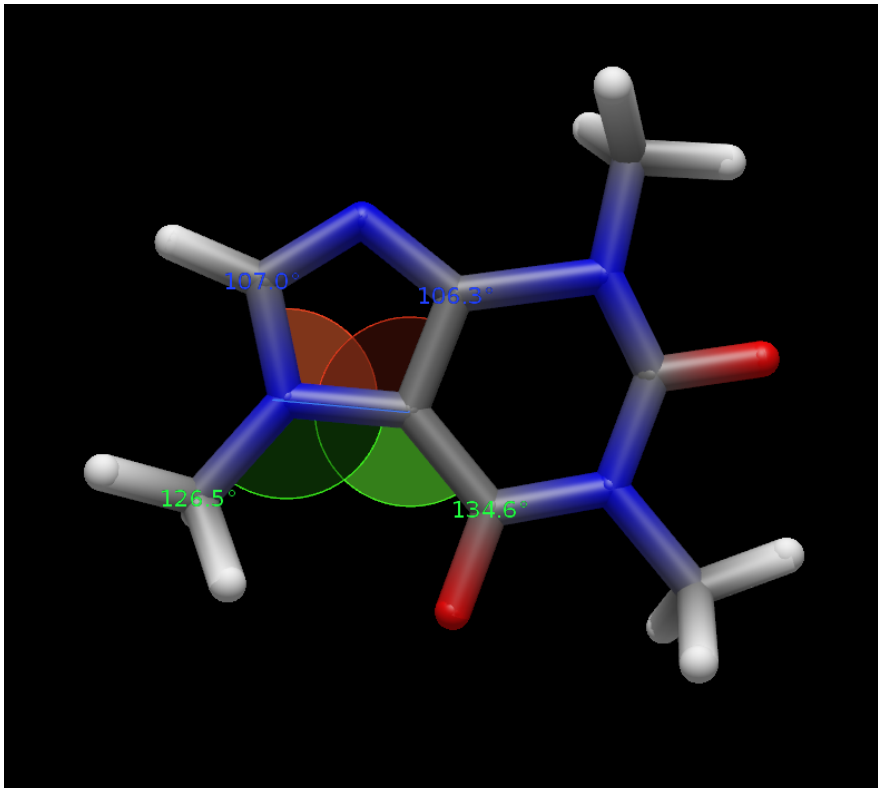


Figure 11: The bond-centric tool, rewritten but retaining most functionality from Avogadro 1.x.

Figure 11 shows the bond-centric tool. The edit tool features an improved periodic table widget that can be resized dynamically. Edit widgets are only enabled if the editable molecule has been used, the general tools are designed to work with both structures. Support for playing molecular trajectories was also added as a tool in order to enable fuller interaction, and support for large trajectory files has been demonstrated.

2.5.10 Scene Plugins

All 3D views of the molecular structure are transformed from input data to objects that can be rendered in the scene graph using scene plugins. These classes have a number of entry points, but the main ones are the process methods that take the molecule to be rendered, and a node in the scene. They populate the node with graphical primitives to be rendered, as shown in the listing below taken from the ball and stick plugin.

```
void BallAndStick::process(const Molecule &molecule,
                           Rendering::GroupNode &node)
{
  GeometryNode *geometry = new GeometryNode;
  node.addChild(geometry);
  // Add a sphere geometry drawable to contain all of the spheres.
  SphereGeometry *spheres = new SphereGeometry;
  spheres->identifier().molecule = &molecule;
  spheres->identifier().type = Rendering::AtomType;
  geometry->addDrawable(spheres);

  for (Index i = 0; i < molecule.atomCount(); ++i) {
    Core::Atom atom = molecule.atom(i);
    unsigned char atomicNumber = atom.atomicNumber();
    if (atomicNumber == 1 && !m_showHydrogens)
      continue;
    const unsigned char *c = Elements::color(atomicNumber);
    Vector3ub color(c[0], c[1], c[2]);
    spheres->addSphere(atom.position3d().cast<float>(), color,
                       static_cast<float>(Elements::radiusVDW(
                          atomicNumber))
                       * 0.3f);
  }

  float bondRadius = 0.1f;
  // Add a cylinder geometry drawable to contain all of the cylinders.
  CylinderGeometry *cylinders = new CylinderGeometry;
  cylinders->identifier().molecule = &molecule;
  cylinders->identifier().type = Rendering::BondType;
  geometry->addDrawable(cylinders);
  for (Index i = 0; i < molecule.bondCount(); ++i) {
    Core::Bond bond = molecule.bond(i);
    if (!m_showHydrogens
        && (bond.atom1().atomicNumber() == 1 || bond.atom2().
            atomicNumber() == 1)) {
      continue;
    }
    Vector3f pos1 = bond.atom1().position3d().cast<float>();
    Vector3f pos2 = bond.atom2().position3d().cast<float>();
    Vector3ub color1(Elements::color(bond.atom1().atomicNumber()));
    Vector3ub color2(Elements::color(bond.atom2().atomicNumber()));
    Vector3f bondVector = pos2 - pos1;
    float bondLength = bondVector.norm();
    bondVector /= bondLength;
    switch (m_multiBonds ? bond.order() : 1) {
    case 3: {
      Vector3f delta = bondVector.unitOrthogonal() * (2.0f *
         bondRadius);
      cylinders->addCylinder(pos1 + delta, bondVector, bondLength,
         bondRadius,
                              color1, color2, i);
      cylinders->addCylinder(pos1 - delta, bondVector, bondLength,
         bondRadius,
                              color1, color2, i);
    }
    default:
    case 1:
      cylinders->addCylinder(pos1, bondVector, bondLength, bondRadius,
                              color1, color2, i);
      break;
    case 2: {
      Vector3f delta = bondVector.unitOrthogonal() * bondRadius;
      cylinders->addCylinder(pos1 + delta, bondVector, bondLength,
         bondRadius,
                              color1, color2, i);
      cylinders->addCylinder(pos1 - delta, bondVector, bondLength,
         bondRadius,
                              color1, color2, i);
    }
    }
  }
}
```

This listing demonstrates how simple the scene API remains, enabling developers of scene plugins to focus on the geometry rather than worrying about OpenGL, global state, batching of draw calls, selection, etc. Settings such as whether to show hydrogen atoms are shown, and configured from the plugins configuration dialog (shown in the application). The full source of the display plugins shows more of the detail, but the above should be enough to demonstrate how simple it is to add new visualizations to Avogadro 2. This approach has been used to process millions of atoms, add them to the scene and render them interactively on consumer-grade laptops running Linux, Mac OS X and Windows.

2.5.11 Extension Plugins

Extensions that are primarily commands executed from the application menus, or additional file formats, utility functionality, etc, were implemented in the extension plugin framework. These extensions can add entries to the menu, which are dynamically created when the application is loaded. They may open dialogs, such as the input generator plugins, calculate new derived quantities, such as the quantum plugins, or add support for new file types, such as the Open Babel plugin and Python IO plugin. Operations on the molecule model, such as bonding, hydrogen addition/removal, and geometry optimization are also possible. Integration with online databases, downloading structures by name and finding similar molecules to the currently shown molecule were all implemented as simple extension plugins that are loaded dynamically. These could easily be extended, or modified to use different data sources, or extend the queries made.

The QTAIM plugin is perhaps the one that contains the largest amount of code at this

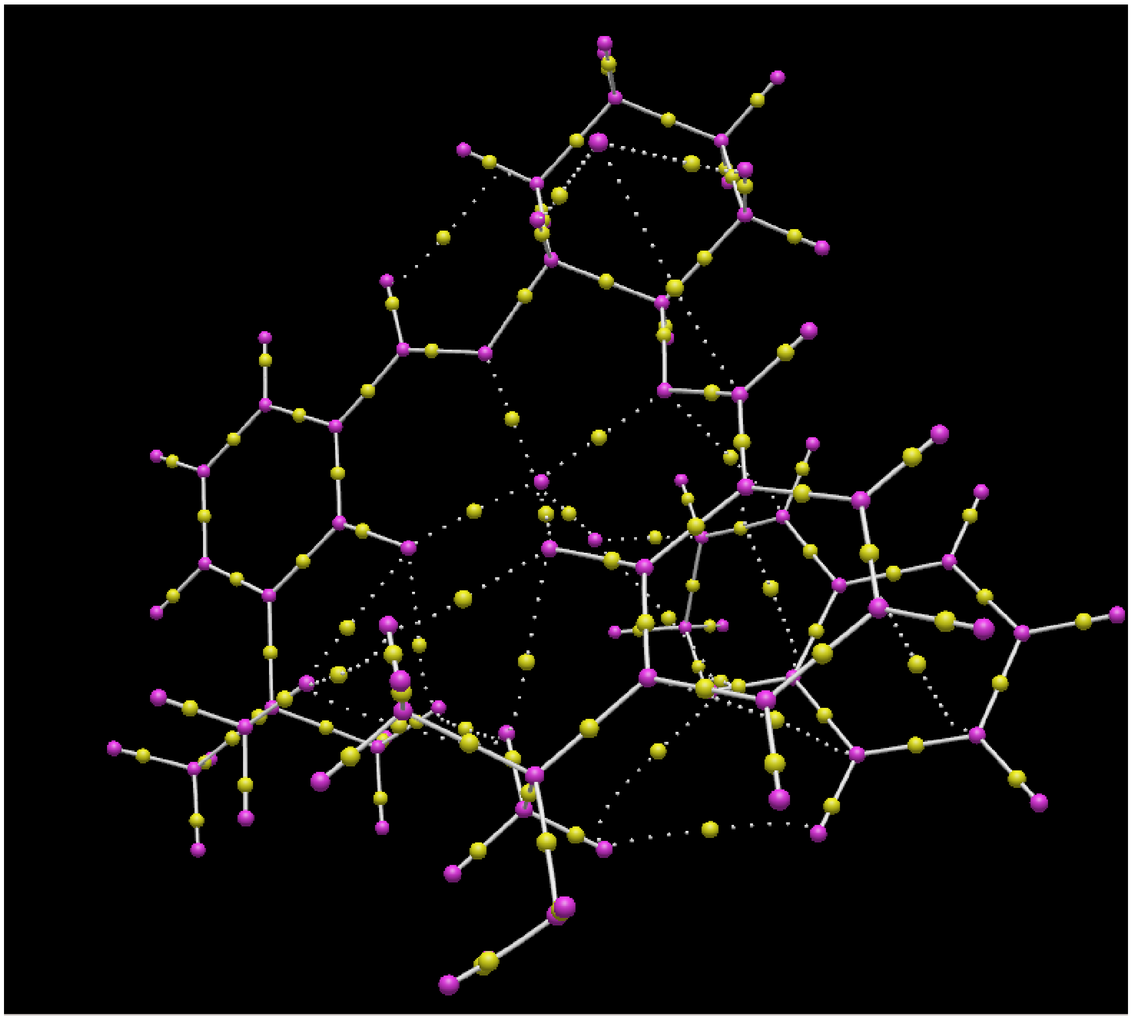


Figure 12: The Avogadro 2 rendering of a wave function file showing QTAIM results.

stage. It was ported from the code contributed to Avogadro 1.x, and could probably be significantly optimized. The basic of the code work, and it is now more fully integrated than was ever before possible. There are also some small pieces of mathematical code, limited to the plugin, that are GPL-licensed, this causes the plugin to be GPL-licensed too due to the copyleft licensing terms at this stage. An example rendering is shown in Figure 12, this component was used in a recent course at the University College London (UCL).

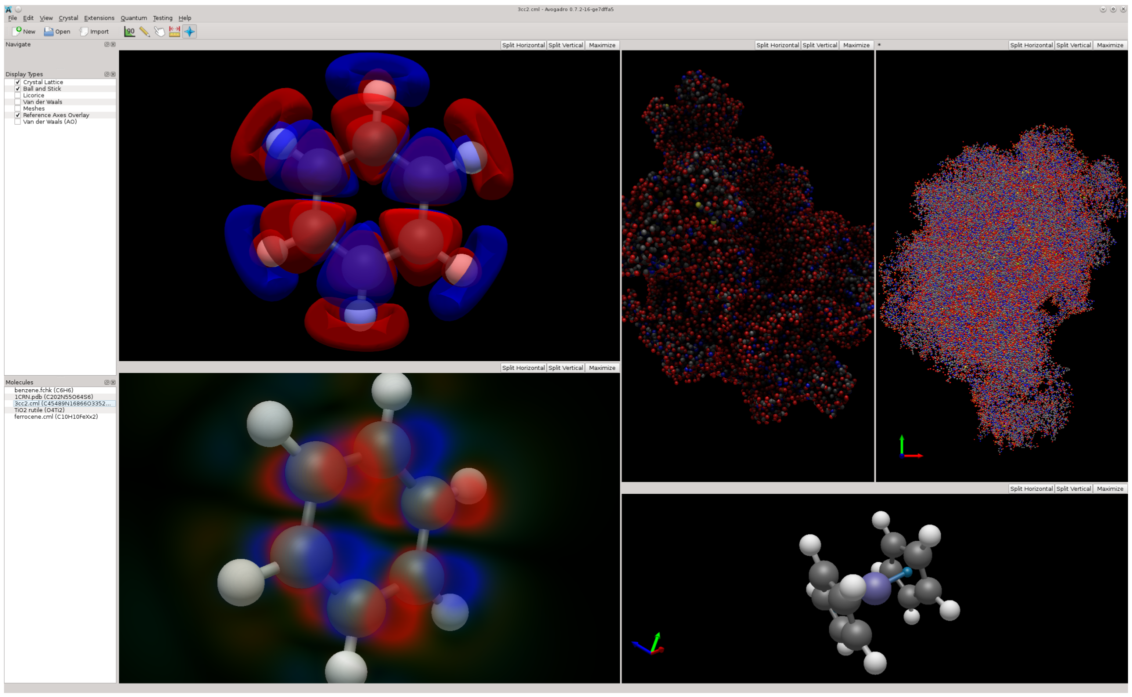
2.5.12 VTK Integration

The Visualization Toolkit (VTK) offers significant visualization and analysis capabilities. One of the primary motivating factors for supporting multiple view widget types was to add a VTK widget to the main application. This has made it possible to offer side-by-side comparisons of electronic structure using simple isosurfaces and volume rendered geometry. In order to offer seamless integration a vtkAvogadroActor class was developed, taking the scene graph based rendering used in the main Avogadro 2 views, and placing them in the rendered 3D view alongside native VTK visualization. This gives a visually appealing solution where the ball and stick rendering is completely identical, with a volume rendering overlaid and the same scene plugins can be reused with no code modifications. Figure 13 shows this capability in the two widgets rendered on the left of the application.

2.6 Avogadro 2 Application

The Avogadro 2 application has also seen quite significant changes, largely due to the significant changes in the Avogadro 2 libraries. The application remains focused on providing an end-user application, ready for use by non-programmers. It makes use of the file format framework, and moves all file loading/saving to background threads. The program supports several command line options, and now offers an RPC server that responds to JSON-RPC 2.0 calls listening on a local names socket.

The application has been ported to use the latest generation of the Qt libraries, with significant changes to the core model to support multiple molecules in a single window, as well as multiple view widgets/widget types, as shown in Figure 13. These changes make it possible to dynamically add a range of display types, seamlessly mixing native Avogadro 2 display widgets with VTK widgets. In the future this framework could easily support yet more widget types, including web-based views seamlessly integrating dynamic web content with the natively rendered views.



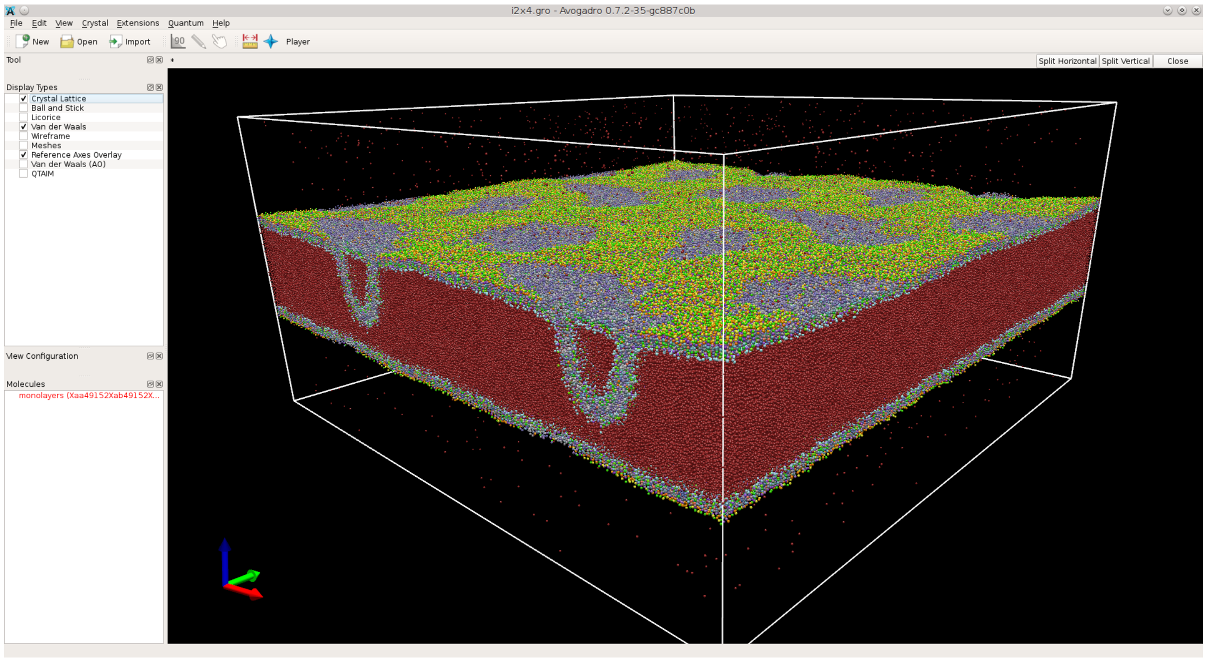
Display Types
- [✓] Crystal Lattice
- [✓] Ball and Stick
- [ ] Licorice
- [ ] Van der Waals
- [ ] Meshes
- [✓] Reference Axes Overlay
- [ ] Van der Waals (AO)

Molecules
- benzene.fchk (C6H6)
- 1CRN.pdb (C202N55O64S6)
- 3cc2.cml (C45489N16866O3352...
- TiO2 rutile (O4Ti2)
- ferrocene.cml (C10H10FeXx2)

Figure 13: The Avogadro 2 application shown displaying different rendering styles of different molecule views.

The application is in a separate repository, ensuring complete independence from the libraries, and acting as a living example of how a custom application can be developed based on the Avogadro 2 libraries. This acts as one of the most complete custom applications, offering the most options along with automated packaging/integration capabilities. The application features integration with MoleQueue, MongoChem and some online chemical databases. The editable molecule widget uses the editable molecule that features native undo/redo support. This application serves as the full-featured demonstration of capabilities in the libraries developed. It can be installed alongside Avogadro 1.x, and while its capabilities exceed those of Avogadro 1.x in many areas, there are still features that have yet to be ported. It has a similar yet distinct icon to highlight the heritage of the project, but the distinct differences present in this rewrite that Avogadro 2 was developed as part of the Phase II SBIR project.

The application has been demonstrated editing new structures, minimizing the structure, generating input for a number of codes, submitting them using MoleQueue, and visualizing the electronic structure of the output. It has also been shown loading structures in excess of 2.8 million atoms on a standard laptop, rendering it interactively, and even showing multiple large and small structures side-by-side. The large structure is shown in Figure 14. Trajectories from molecular dynamics calculations can be loaded, and animated in the main interface. Support for periodic structures is also present, along with client-server capabilities that enable interaction with large data on remote systems using a server-resident process.


i2x4.gro - Avogadro 0.7.2-35-gc887c0b
File Edit View Crystal Extensions Quantum Help
New Open Import Player
Tool
Split Horizontal Split Vertical Close
Display Types
View Configuration
Molecules

[✓] Crystal Lattice
[ ] Ball and Stick
[ ] Licorice
[✓] Van der Waals
[ ] Wireframe
[ ] Meshes
[✓] Reference Axes Overlay
[ ] Van der Waals (AO)
[ ] QTAIM

monolayers (Xaa49152Xab49152X...

Figure 14: The Avogadro 2 application displaying a GROMACS structure, kindly shared by Peter Tieleman, with over 2.8 million atoms.

This application represents a significant step forward in capabilities, positioned to make a significant impact on the field. It is easily extensible with modest levels of expertise, and can handle very large and very small structures equally well. The layout of the libraries, plugin interfaces and licensing make it especially amenable to use in various DoD projects where custom functionality can be implemented for specialized application areas.

2.7 MongoChem

The MongoChem application is developed using C++, Qt, MongoDB, VTK, and the Avogadro 2 libraries. Some of its cheminformatics functionality coming from Open Babel and Chemkit. This application is focused on facilitating the deposition of chemical data in a scalable database, adding various properties to the molecules, and facilitating the dynamic visualization and analysis of aggregate data. It has been generalized to support connections to different MongoDB database servers, including the use of shared servers, in the cases where very large databases are required.

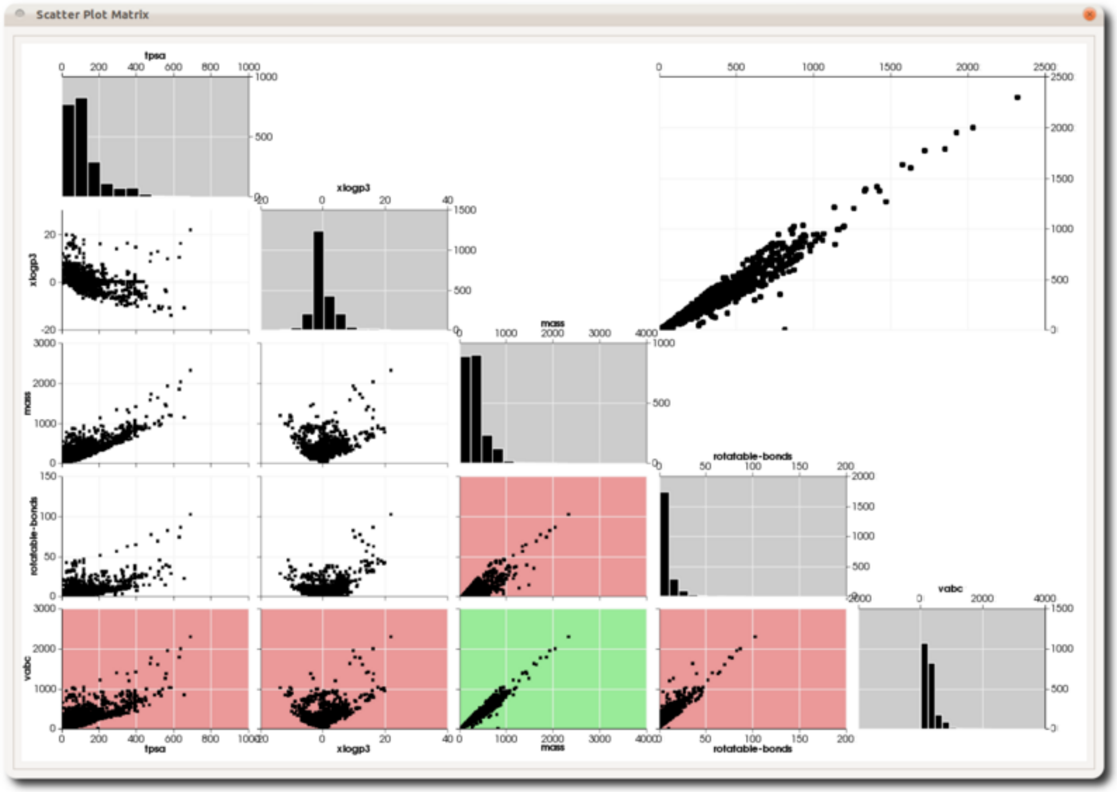


Figure 15: The scatter plot matrix view showing multiple .

The range of charts available has been extended to include simple histograms and scatter plots, through to multivariate visualization techniques such as parallel coordinates and scatter plot matrices (which combine scatter plots for multiple dimensions, along with population histograms for each variable and linked selections, shown in Figure 15). The charts make use of VTK's selection linking functionality that enables users to make and visualize subsets of the data in many different views and representations, as shown in Figure 16.

The use of molecule fingerprinting techniques gives the database the ability to be searched by similarity to a desired structure, as well as enabling queries on chemical name, tag, and other properties. These results can then be viewed in the different data display views available in order further inspect the selected subset of data or do further calculations/export. This enables integration of independently developed software, such as that used to generate QSAR data, into the framework while enabling scientists to make use of the analytical

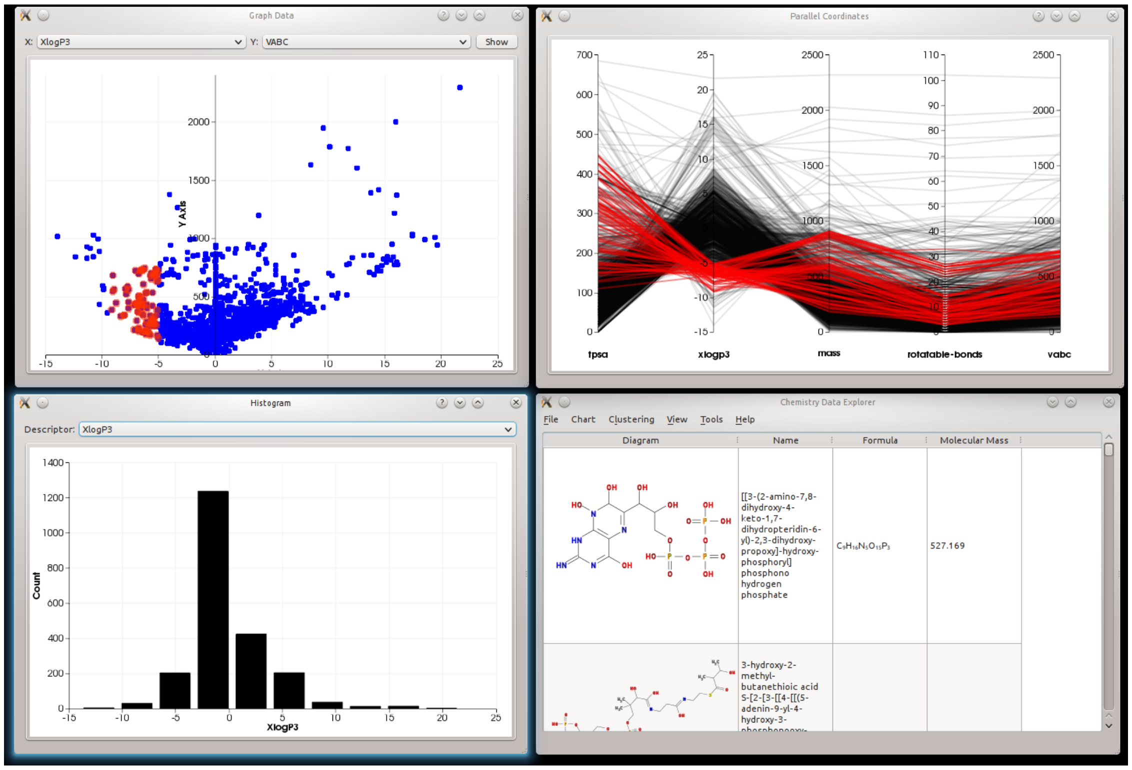


Diagram	Name	Formula	Molecular Mass
	[[3-(2-amino-7,8-dihydroxy-4-keto-1,7-dihydropteridin-6-yl)-2,3-dihydroxy-propoxy]-hydroxy-phosphoryl] phosphono hydrogen phosphate	$C_9H_{16}N_5O_{15}P_3$	527.169
	3-hydroxy-2-methyl-butanethioic acid S-[2-[3-[[4-[[(5-adenin-9-yl-4-hydroxy-3-		

Figure 16: Linked selection in several charts and the table view in MongoChem.

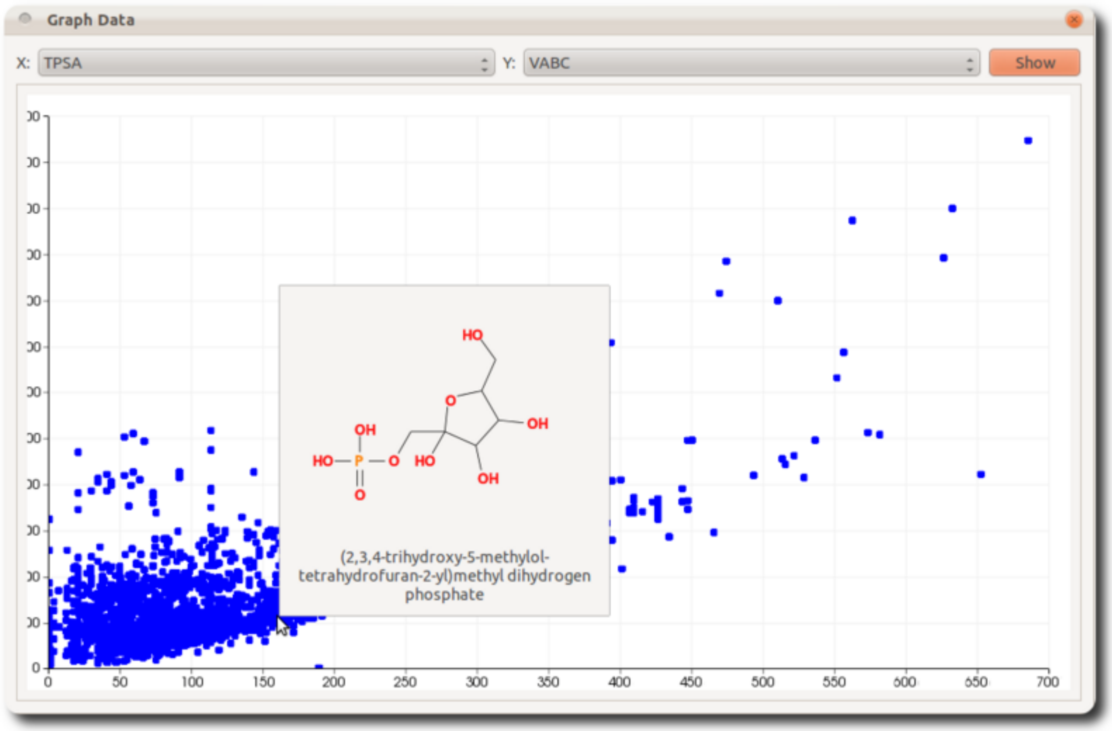
capabilities of the application.

Many of the charts in the application feature intelligent tooltips that display the 2D structure along with the IUPAC name, as shown in Figure 17. Network connectivity views based both on fingerprint and structural similarity enable users to view the overall relatedness of compounds in the database. The K-means clustering view displays descriptor values using 3D charts and groups them based on similarity (see Figure 18); the view features interaction (panning, zooming, etc), and the clustering parameters can be modified.

Large data sets can be imported using simple Python scripts, or with graphical tools such as the CSV importer in the application. In addition to the desktop functionality, a prototype web application has been developed which shows the data from the same MongoDB store using modern web techniques coupled with the Python-based server-side frameworks and VTKWeb framework to give any user read-only access to the data, and the ability to visualize basic 2D and 3D structure (with interaction for the 3D visualization).

In addition to the charts and table view a detailed dialog is available once a single molecule has been selected, as shown in Figure 19; this gives further details on the structure, such as InChI and SMILES strings. The detailed table views enable the export of structures, or to directly open the structure in Avogadro. Structures can have multiple tags that are searchable, and annotations can be saved with notes relevant to the structure.

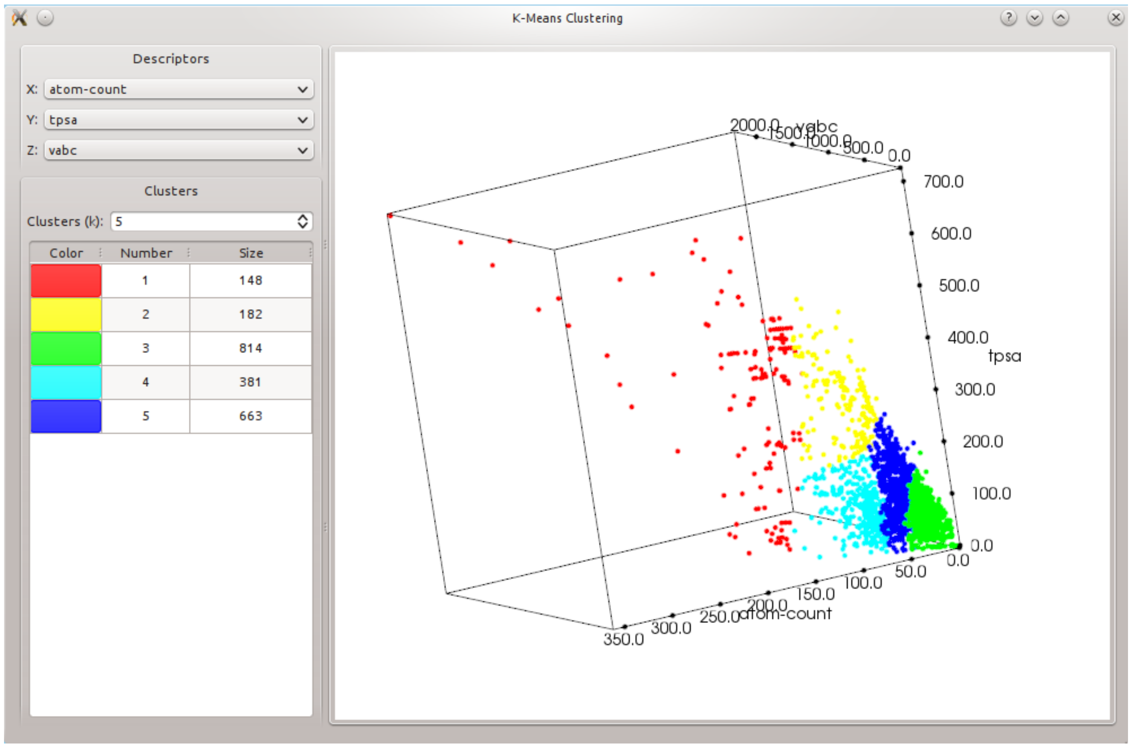
A collaboration with the Aspuru-Guzik group at Harvard University has also made available a large number of electronic structure calculations. The data set is about 0.5 PB in size, and an initial 4 TB sample has been duplicated for testing and performance benchmarking. The data set is interesting for the MongoChem application as it includes a large number of



X: TPSA Y: VABC

(2,3,4-trihydroxy-5-methylol-tetrahydrofuran-2-yl)methyl dihydrogen phosphate

Figure 17: Custom tooltips in scatter plots displaying the 2D structure and IUPAC name of the point under the cursor.

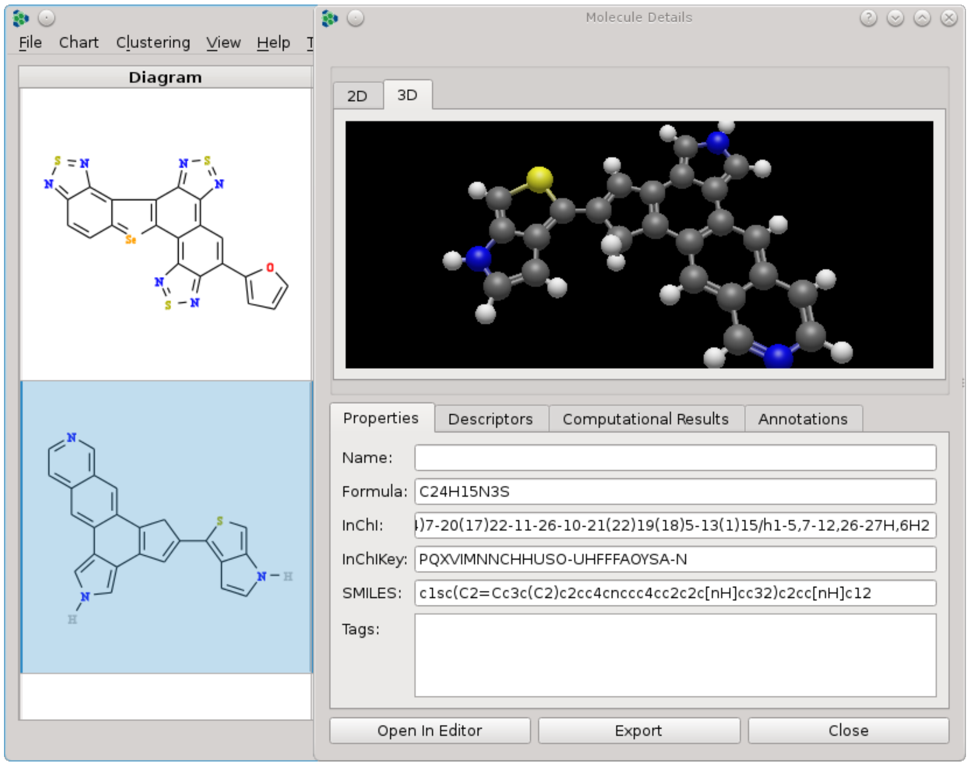


X: atom-count
Y: tpsa
Z: vabc

Clusters (k): 5

Color	Number	Size
🟥	1	148
🟨	2	182
🟩	3	814
🟦	4	381
🟦	5	663

Figure 18: K-means clustering view in MongoChem.



Name:	
Formula:	C24H15N3S
InChI:	)7-20(17)22-11-26-10-21(22)19(18)5-13(1)15/h1-5,7-12,26-27H,6H2
InChIKey:	PQXVIMNNCHHUSO-UHFFFAOYSA-N
SMILES:	c1sc(C2=Cc3c(C2)c2cc4cnccc4cc2c2c[nH]cc32)c2cc[nH]c12
Tags:	

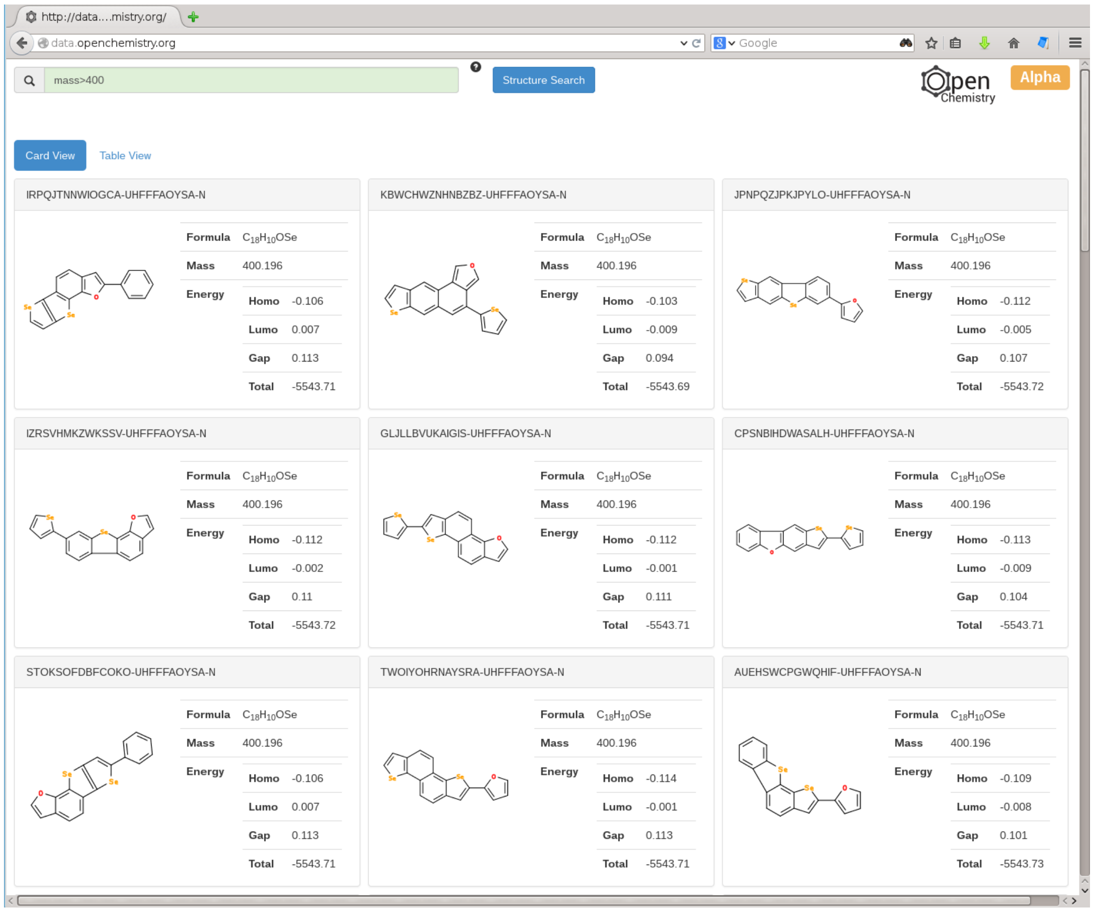
Figure 19: The detailed dialog in MongoChem for a molecular structure.

small molecule structures that are candidates for organic solar cell materials,[20] with multiple calculations per structure using different levels of theory and calculation types. A modified version of Q-Chem was used to perform the calculations on the World Community Grid.[21]

There is a live demo available at data.openchemistry.org that contains a snapshot of the Clean Energy Project's data, showing a customized web view, searchable database, and capability to both display and download 3D structures. The client code is largely JavaScript/HTML5, using simple RESTful APIs to retrieve data—these same APIs could be used by desktop applications. The data available on the site can also be displayed and edited in the desktop MongoChem application.

Figure 20 shows a capture of the card view offered by the site, summarizing some of the calculation details. Clicking on any card opens up a detailed view, which also has an interactive 3D view of the structure. The web demo uses an in-memory fingerprint database in order to accelerate similarity and substructure searches, which remain one of the most important queries that are quite poorly supported by the underlying MongoDB database server technology.

During the course of development it became clear that the MongoDB C++ client libraries do not offer a stable API, and the MongoDB facilities need to be augmented with more capabilities on the server-side such as the in-memory structure search capabilities. A more complete solution would implement simple authentication, access control, and acceleration capabilities on the server-side with a thin interface through to MongoDB. This would make the offering more flexible and extensible, but would require more investment in order to fully realize. The solution developed remains viable, even for collections of millions of



	Formula	Mass	Energy: Homo	Lumo	Gap	Total
IRPQJTNNWIOGCA-UHFFFAOYSA-N	$C_{18}H_{10}OSe$	400.196	-0.106	0.007	0.113	-5543.71
KBWCHWZNHNBZBZ-UHFFFAOYSA-N	$C_{18}H_{10}OSe$	400.196	-0.103	-0.009	0.094	-5543.69
JPNPQZJPKJPYLO-UHFFFAOYSA-N	$C_{18}H_{10}OSe$	400.196	-0.112	-0.005	0.107	-5543.72
IZRSVHMKZWKSSV-UHFFFAOYSA-N	$C_{18}H_{10}OSe$	400.196	-0.112	-0.002	0.11	-5543.72
GLJLLBVUKAIGIS-UHFFFAOYSA-N	$C_{18}H_{10}OSe$	400.196	-0.112	-0.001	0.111	-5543.71
CPSNBIHDWASALH-UHFFFAOYSA-N	$C_{18}H_{10}OSe$	400.196	-0.113	-0.009	0.104	-5543.71
STOKSOFDBFCOKO-UHFFFAOYSA-N	$C_{18}H_{10}OSe$	400.196	-0.106	0.007	0.113	-5543.71
TWOIYOHRNAYSRA-UHFFFAOYSA-N	$C_{18}H_{10}OSe$	400.196	-0.114	-0.001	0.113	-5543.71
AUEHSWCPGWQHIF-UHFFFAOYSA-N	$C_{18}H_{10}OSe$	400.196	-0.109	-0.008	0.101	-5543.73

Figure 20: The MongoChemWeb live demo showing data from the Clean Energy Project.

molecules, but has some bottlenecks in search capabilities and could only be deployed on trusted networks at this stage (with an Internet facing web component optionally, such as that prototyped).

The application features integration with MoleQueue, where subsets can use the input generator framework from Avogadro 2 and submit computational jobs. It also features integration with Avogadro 2, offering the capability to open structures in the running Avogadro 2, and show similar structures in its database when initiated from Avogadro 2. These local RPC calls leverage the framework developed in MoleQueue, and more calls can easily be added in future. The application makes use of Avogadro 2 rendering widgets, file formats, input gecommitnerators, with scope to increase the level of interaction in future versions.

2.8 MoleQueue

The MoleQueue application is a C++ Qt application developed to provide an abstraction to local and remote computational resources. Its functionality is not chemistry-specific, but it is necessary in order to remove many of the barriers encountered by users attempting to make use of computational chemistry applications on both the desktop and remote computational resources.

The project provides two components: a system-tray resident application where remote and local computational resources are configured to act as a local job dispatch server, and a client that uses a remote procedure call (RPC) API to stage and submit computational jobs. The RPC API uses a data structure called JSON (JavaScript Object Notation) to carry data in a language and architecture independent fashion. This format was chosen due to the vast array of implementations in virtually every programming language. The JSON RPC 2.0 specification builds upon the JSON data format in order to provide a cross-platform, device-independent RPC API that can easily be implemented in any language desired. Finally, the local socket transport has been chosen due to the security considerations that require local sockets to follow the same permission model as files on every operating system supported - only users with access to the users files on the local system can access a local socket to submit jobs.

It is possible to add further transports, but the communication protocol will remain very simple and easy to implement. A C++ Qt client library is provided, along with some reference implementations for submitting jobs using the Python language. The JSON-RPC 2.0 specification consists of a protocol identifier string-value pair, request, response, notification, and error messages. Requests consist of a JSON-RPC message that contains a method key with a corresponding method name and a params object that contains any method call parameters necessary to complete the request. The client will receive a reply or an error with an id matching that of the request.

Some simple examples, such as requesting a list of queues and their programs is as simple as:

```
{
  "jsonrpc": "2.0",
  "method": "listQueues",
  "id": 42
}
```

The response to this request might look something like this:

```
{
    "jsonrpc": "2.0",
    "result": {
        "Diamond": [
            "GAMESS",
            "MOPAC",
            "Gaussian",
            "NWChem"
        ],
        "Garnet": [
            "GAMESS",
            "MOPAC",
            "Gaussian",
            "NWChem"
        ],
        "Local": [
            "GAMESS",
            "MOPAC",
            "Gaussian",
            "NWChem"
        ]
    },
    "id": 42
}
```

A slightly more complex RPC call to submit a job using MoleQueue would look as follows:

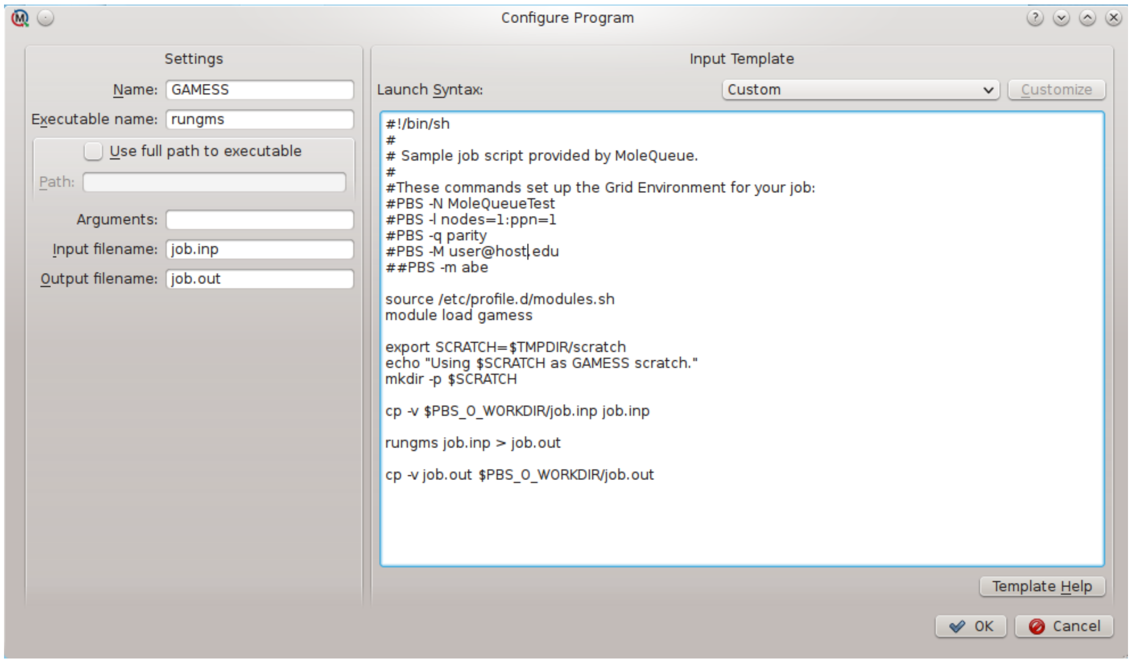
```
{
  "jsonrpc": "2.0",
  "method": "submitJob",
  "params": {
    "queue": "Garnet",
    "program": "GAMESS",
    "description": "B3LYP H2O optimization",
    "inputFile": {
      "filename": "job.inp",
      "contents": "Full contents of input file.\nWill be created in
         the working tree."
    }
  },
  "id": 23
}
```

This submits a job to the remote queue named "Garnet," with the program named "GAMESS". The description is the string that will show up in the MoleQueue user interface, and the input file is specified by either a full path or a file name and contents string. The response for a successful submission looks something like the following:

```
{
  "jsonrpc": "2.0",
  "result": {
    "moleQueueId": 17,
    "workingDirectory": "/tmp/MoleQueue/17/"
  },
  "id": 23
}
```

This response object gives a long-lived identifier for the job, "moleQueueId," along with the working directory where all files will be staged. If there was an error then an error response will be generated (with an "id" matching that of the request) and an appropriate error, such as queue or program does not exist. Once a job is submitted, notifications are sent when the job state changes; for example, from submitted to running, error, completed etc. Each of the notifications carries the moleQueueId, along with the notification of the job-state change. It is then possible to act upon these changes. There are also RPC methods to query jobs, which can give access to remote job identifiers or to cancel an already submitted job.

The MoleQueue application runs in the system tray, and provides a graphical interface to define remote queues, how to execute programs on those remote queues, and act as an interface for event logging/current job status on all remote systems to which MoleQueue has submitted jobs. Figure 21 shows the program configuration dialog; those familiar with PBS submission scripts should recognize the parameters. A simple keyword substitution is used to replace keywords specified in the template with job-specific settings. This file will be constructed upon job submission and uploaded to the remote system. It will submitted to the batch scheduler, typically using the `qsub` command or its equivalent. Queue/program settings can be imported and exported, facilitating the easy set up of queues across sites if system administrators distribute relevant files with their submission criteria.



Name: GAMESS
Executable name: rungms
Use full path to executable
Path:
Arguments:
Input filename: job.inp
Output filename: job.out
Launch Syntax: Custom

```
#!/bin/sh
#
# Sample job script provided by MoleQueue.
#
#These commands set up the Grid Environment for your job:
#PBS -N MoleQueueTest
#PBS -l nodes=1:ppn=1
#PBS -q parity
#PBS -M user@host.edu
##PBS -m abe

source /etc/profile.d/modules.sh
module load gamess

export SCRATCH=$TMPDIR/scratch
echo "Using $SCRATCH as GAMESS scratch."
mkdir -p $SCRATCH

cp -v $PBS_O_WORKDIR/job.inp job.inp

rungms job.inp > job.out

cp -v job.out $PBS_O_WORKDIR/job.out
```

Figure 21: The program configuration dialog in MoleQueue.

In order to be useful on as many high performance computing resources as possible, it was necessary to implement several secure transport methods. The first of which is SSH (secure shell) which is an industry standard employed by many of the world's largest supercomputers to provide access to computational resources. The MoleQueue application can call a specified SSH command line client, or make use of the libssh2 library. The main reason for providing support for both is the lack of Kerberos support in libssh2 and the custom patches applied by some sites to SSH clients in order to support different challenge-response authentication techniques. Once a transport has been chosen to support authentication, command dispatch and file transfer, it is then necessary to support the batch scheduling systems in use on the HPC resource—primarily Sun Grid Engine, PBS, and SLURM.

Support for SSH with major batch scheduling systems enables MoleQueue to support a large range of supercomputers and cloud resources (including Amazon's EC2 when used with StarCluster to deploy a Sun Grid Engine cluster on demand). The MoleQueue backends are abstracted in such a way to allow for many compute resource backends to be added. In order to take advantage of the HPC resources provided to ERDC researchers, integration with the ezHPC platform was also necessary. This has been accomplished using two approaches, the first being the more generic SSH transport coupled with PBS, and the second being the use of the UIT SOAP-like API provided for automated use of HPC resources over an HTTPS transport.

Unfortunately the UIT documentation does not provide enough detail in some instances, and is incorrect in a few places. Only parts of the HPC access have been modeled in

SOAP, and so it is necessary to manually parse many of the responses and match them to the raw PBS response and error codes. Kitware developers have spent significant time using a combination of traditional SOAP implementations, following documentation, trial-and-error, and direct communication with ezHPC UIT support staff in order to provide a working implementation. We have reported several issue/bugs to the UIT support address, and will continue to work with them to resolve these issues. At this stage, the vast majority of the core functionality is implemented, but there are error conditions and event sequences where the API tokens and data flow remain unclear. Due to the MoleQueue abstraction, it is possible to use a UIT queue or a direct SSH/PBS queue to dispatch, monitor, and retrieve results.

2.9 VTK

The Visualization Toolkit (VTK) is a large, open source, cross-platform toolkit for data processing and visualization. It has many specialized classes for data processing, informatics, mathematics, data handling, computational fluid dynamics, geospatial visualization, medical imaging, charting, volume rendering, and other areas. One area that has not been added to VTK until now was support for chemical data structures and visualization. Many projects have used VTK for molecular rendering and visualization, but have had to extend it in their own applications and have not been able to benefit from built-in support.

VTK has been extended with a dedicated chemistry module that provides hardware-accelerated visualization making use of advanced support for glyphs in order to get maximum performance. Support for standard atom color schemes and the standard molecular representations have been added. The readers have been augmented to read in secondary protein structure and use the ribbon rendering representations expected for larger biological structures. Additional file format support has been added, along with optimizations for larger structures and interaction.

This means that applications using VTK can benefit from built-in support for chemical structure visualization, along with all the other visualization techniques and data processing code present in the library. The 2D visualization techniques have also been extended in order to better support applications in chemistry, such as custom tooltip support (enabling 2D structures to be displayed in tooltips) and support for multidimensional visualization and selection. Various additional chart types and support for seamless 2D-to-3D chart transitions offer more immersive visualization and analysis environments in Open Chemistry applications. The client-server applications using VTK, such as ParaView and ParaViewWeb, can also benefit from this new functionality and be leveraged in chemical applications.

3 Conclusions

The project has achieved all core goals, and has prompted several new collaborations that are resulting in wider impacts in the chemistry community. The three Open Chemistry applications (MongoChem, MoleQueue, and Avogadro 2) are available in both source and binary form for all major platforms. The JSON-RPC 2.0 inter-process communication APIs and common data structures have been developed to facilitate seamless communication between the loosely coupled components.

The Avogadro 2 libraries and application have been demonstrated on small molecules involving expensive quantum calculations through to large molecular dynamics simulations containing over 2.8 million atoms. The MongoChem application also saw similar success, being useful for analyzing small molecule collections through to some of the largest available—the Harvard Clean Energy project with millions of unique molecules and hundreds of millions of quantum calculations.

The MoleQueue project gained some additional funding from an ERDC PETTT project for use in another application domain, and several National Labs have expressed an interest in integrating MoleQueue in their applications, including a climate modeling project demo. This interest has led to successful integration with several supercomputers and schedulers (in addition to UIT/ezHPC, Sun Grid Engine and PBS as part of this project). Collaborations with EMSL's NWChem project and Harvard's clean energy project have provided multiple viewpoints on current opportunities in the area for powerful desktop applications in preparing input, integrating with HPC resources and applying cheminformatics techniques to the indexing and analysis of large numbers of quantum calculations/small molecules. Collaborations with the Peter Tieleman group, and discussions with researchers based at Sandia and Los Alamos National Laboratories have offered a view of work taking place in large molecular dynamics simulations containing millions of atoms.

The projects have gained a significant feature set, and offer unique capabilities. Some initial funding has been obtained to explore multiscale modeling approaches, using the results of this project as one of the major foundations. Other nascent collaborations are exploring areas as diverse as heavy element compound calculations, large-scale materials simulations, biological system and drug delivery systems. We remain committed to the approach taken, and will continue to develop the project. All projects were successfully migrated to Qt 5 recently, ensuring their viability in the coming years. The software libraries, applications and patterns developed position Kitware well to become a major force in this and related areas.

References

[1] Qt. Online (June 2010). http://qt.nokia.com/.

[2] Cmake. Online (June 2010). http://www.cmake.org/.

[3] *Mastering CMake*. Kitware, Inc., 5th edition (2010).

[4] Vtk. Online (June 2010). http://www.vtk.org/.

[5] Schroeder, W., Martin, K. and Lorensen, B. *An Object Oriented Approach to 3D Graphics*. Kitware, Inc., 4th edition (2004).

[6] *The VTK User's Guide*. Kitware, Inc., 11th edition (2010).

[7] Open babel. Online (June 2010). http://www.openbabel.org/.

[8] OBoyle, N., Banck, M., James, C., Morley, C., Vandermeersch, T. and Hutchison, G. *Journal of Cheminformatics*, **3**, (2011) 1–14. ISSN 1758-2946. http://dx.doi.org/10.1186/1758-2946-3-33.

[9] Rdkit. Online (June 2010). http://www.rdkit.org/.

[10] Open chemistry. Online (November 2012). http://www.openchemistry.org/.

[11] Code review, topic branches and vtk. Online (April 2012). http://www.kitware.com/source/home/post/62.

[12] Gerrit. Online (November 2012). http://code.google.com/p/gerrit/.

[13] Cdash@home. Online (October 2010). http://www.kitware.com/source/home/post/21.

[14] Json specification. Online (November 2012). http://www.json.org/.

[15] Mongodb. Online (November 2012). http://www.mongodb.org/.

[16] Bson specification. Online (November 2012). http://bsonspec.org/.

[17] Murray-Rust, P. and Rzepa, H. S. *Chemistry Intl.*, **4**(24), (2002) 9–13.

[18] Avogadro. Online (June 2010). http://avogadro.openmolecules.net/.

[19] Hanwell, M. D., Curtis, D. E., Lonie, D. C., Vandermeersch, T., Zurek, E. and Hutchison, G. R. *Journal of Cheminformatics*, **4**(1), (2012) 17.

[20] The clean energy project. Online (November 2012). http://cleanenergy.harvard.edu/.

[21] World community grid. Online (November 2012). http://www.worldcommunitygrid.org/.