2009

Development of High Performance Scientific Components for Interoperability of Computing Packages

Teena Pratap Gulabani  
*Iowa State University*

Masha Sosonkina  
*Iowa State University*

Mark S. Gordon  
*Iowa State University*, mgordon@iastate.edu

Curtis L. Janssen  
*Sandia National Laboratories*

Joseph Kenny  
*Sandia National Laboratories*

*See next page for additional authors*

Follow this and additional works at: [http://lib.dr.iastate.edu/ameslab_conf](http://lib.dr.iastate.edu/ameslab_conf)  
Part of the [Chemistry Commons](http://lib.dr.iastate.edu/ameslab_conf), and the [Computer Sciences Commons](http://lib.dr.iastate.edu/ameslab_conf)

**Recommended Citation**  
[http://lib.dr.iastate.edu/ameslab_conf/91](http://lib.dr.iastate.edu/ameslab_conf/91)

This Article is brought to you for free and open access by the Ames Laboratory at Digital Repository @ Iowa State University. It has been accepted for inclusion in Ames Laboratory Conference Papers, Posters, and Presentations by an authorized administrator of Digital Repository @ Iowa State University. For more information, please contact digirep@iastate.edu.
Authors
Teena Pratap Gulabani, Masha Sosonkina, Mark S. Gordon, Curtis L. Janssen, Joseph Kenny, Heather Marie Netzloff, and Theresa Lynn Windus
Development of High Performance Scientific Components for Interoperability of Computing Packages

Teena P. Gulabani, Masha Sosonkina, Mark S. Gordon, Curtis L. Janssen, Joseph P. Kenny, Heather Netzloff, Theresa L. Windus

Keywords: Interoperability, NWChem, Common Component Architecture, QM/MM Modeling, TAU

Abstract
Three major high performance quantum chemistry computational packages, NWChem, GAMESS and MPQC have been developed by different research efforts following different design patterns. The goal is to achieve interoperability among these packages by overcoming the challenges caused by the different communication patterns and software design of each of these packages. Developing a chemistry algorithm is a time consuming process; integration of large quantum chemistry packages will allow resource sharing and thus avoid reinvention of the wheel. Creating connections between these incompatible packages is the major motivation of our work. We achieve this interoperability by bringing the benefits of Component Based Software Engineering through a plug-and-play component framework called Common Component Architecture (CCA). In this paper, we present a strategy and process used for interfacing two widely used and important computational chemistry methodologies: Quantum Mechanics and Molecular Mechanics. This paper also demonstrates the performance evaluation of these CCA compliant components to show the feasibility of the proposed approach and finally discusses the current research issues.

1. INTRODUCTION
Software reuse is the process of building systems using existing software rather than building the systems from scratch. Quantum Chemistry packages like NWChem [1, 2], GAMESS [3] and MPQC [4] are developed to perform high-performance scientific simulations. These packages are developed and maintained by different research scientists. It is difficult for a single research group to effectively develop solutions for all of the methods one would desire. Also, since each research group needs some similar capability, it leads to duplication of efforts. When a new capability is to be added to a package, it is nice to have a gateway via which packages can reuse existing tested capabilities. Additionally, research groups often optimize their software for a particular hardware. Since the different groups optimize for different hardware, the different algorithms for the same functionality may run better on different platforms. Being able to use the best algorithm for a particular platform increases the overall throughput of the science. This has the potential to significantly improve our ability to construct effective software systems. Commercial component based software engineering practices such as COM [5], EJB [6] and CORBA [7] exists in the market, which present an approach for managing the increasing complexity of business packages. In our research work, we use a similar framework known as Common Component Architecture (CCA) [8], which is targeted at high-performance scientific development. CCA allows research scientists to create components, which can be used by different research groups through well-defined interfaces. As a result of this synergy, rapid development is possible while avoiding redundant efforts.

The remainder of this paper is structured as follows: Section 2 gives a brief description of Quantum Mechanics and Molecular Mechanics (QM/MM) [9]. Section 3 discusses the three computational packages, NWChem, MPQC and GAMESS. It also gives a brief review of CCA - one of the pillars of our work. Section 4 describes the design and implementation details of the QM/MM component model. Section 5 discusses how interoperability is possible among different computational packages. Section 6 gives a review of performance analysis of the QM/MM component model. Finally, section 7 states conclusions and current research issues.

2. BACKGROUND
The rapid increase in computer speed and hardware technology advancement has made quantum chemistry a practical tool for chemists in various branches of chemistry, such as organic, inorganic, analytical, and physical. However, while QM methods can treat chemical reactions accurately, they are quite expensive for very large molecules. On the other hand, MM methods can treat very large molecules but are not well suited for chemical reactions. To deal with the chemical reactions in very large systems, a combined QM/MM method is used where quantum mechanics calculation is embedded in a molecular mechanics model. In this, quantum mechanics can be used to treat the part of the system affected by the reaction, and
molecular mechanics to treat the rest of the chemical environment. For example, for a reaction in solution, one treats the reacting solute molecules and the first solvent shell using QM and the surrounding solvent molecules using MM.

![Figure 1. Regions involved in QM/MM energy calculation of tripeptide alanine-serine-alanine (asa)](image)

The challenge is to find an appropriate boundary region between QM and MM regions and to accurately describe the physics of the boundary region. This is critical because a selection of an inappropriate boundary will produce poor end results. There are many choices for the physics of the boundary conditions. This work will focus on two models. The first is a model where molecules are weakly interacting with one another (i.e. not bonded to one another) and only the charges or electrostatics of the MM region need to be taken into account in the QM calculation. In the second model, the border contains atoms that are bonded to one another. Figure 1 shows both QM and MM regions of tripeptide alanine-serine-alanine. In this model, the quantum calculation will need to substitute the rest of the MM region by hydrogen or fluorine like caps where bonds are broken as well as the electrostatic charges for the rest of the MM region. In both models, the electrostatic interactions are included in the QM computation so that the QM wave function feels the effects of the environment of the MM atoms – allowing the reaction to be modified based on the environment. Overall, the QM/MM approach combines the strength of both QM (accuracy) and MM (speed) packages.

3. QUANTUM CHEMISTRY PACKAGES AND COMPONENT BASED SOFTWARE DESIGN

Three important high performance quantum chemistry computational packages are NWChem, GAMESS and MPQC. Each of these packages is developed by various research scientists across the world and has its design and development history. Only a very brief introduction for each software package is given here. Full details are available in references [1, 2, 3, 4].

NWChem, primarily written in Fortran77 (with a few functions written in C) is a computational chemistry software package developed at Pacific Northwest National Laboratory. It is designed to run on high performance parallel supercomputers as well as on standalone workstations. NWChem uses Global Arrays (GA) [10] that is a tool designed to complement the message passing-programming model. GA provides a shared memory interface for distributed memory computers. NWChem has a modular architecture and follows object oriented principles. It is used to perform quantum mechanics, molecular mechanics and combined QM/MM simulations.

GAMESS, mainly written in Fortran77 (along with some C), is a flexible ab initio electronic structure program that can compute many wave functions and properties of interest to chemists. GAMESS uses Distributed Data Interface (DDI) as the message-passing layer to support the parallel execution of GAMESS.

MPQC is an ab initio computational chemistry software package written in C++ (with a few C functions) that also has a full suite of wave function methods available. MPQC uses Message Passing Interface (MPI) as the message-passing layer to support parallel execution.

All these packages have their strengths and weaknesses. The goal of our research is to provide an easy way to utilize the best of all the packages without imposing a significant performance penalty. We have developed several high performance CCA compliant components for computational packages including QM components to calculate energies, gradients and Hessians for optimization and to compute integrals necessary for most QM methods [11, 12]. In this paper, we focus on the development of components to perform QM/MM calculations.

CCA helps in managing software complexity by bringing the benefits of a lightweight component system to high performance computing in science. CCA is built on object-oriented principles such as encapsulation, abstraction, data hiding, interfaces and modularization and adds the flexibility of building applications in a plug-and-play fashion. Interfaces define the required behavior for interaction between different components. Components communicate with one another via ports. A component can have two types of ports: provides port (i.e. component implement the functionality, which can be used by other components) and uses port (component uses the functionalities provided by another component). Ccaffeine[13] is the framework used in this work for connecting the ports and for running componentized applications. Thus, CCA allows us to instantiate components dynamically that have well-defined generic interfaces and run them by connecting the appropriate well-defined ports.

To deal with the language differences between components, this project uses SIDL (Scientific Interface Definition Language) and the Babel code [14] to generate inter-language bindings. In SIDL, a class that implements gov.cca.Component interface represents a CCA component. Additionally, it also implements interfaces corresponding to any CCA ports that it wishes to provide to other components).
4. DESIGN AND IMPLEMENTATION DETAILS
Creating a QM/MM component model for an existing program or functionality is a four-step process:

- Deciding the flow of the QM/MM run and how to pass information accordingly.
- Specifying generic interfaces through a SIDL specification
- Implementing interfaces (often by writing wrappers to the native library functions)
- Providing and using ports in components that enable communication of data between the components.

The first two steps are critical in the design phase. Writing interfaces involves a lot of assessment since the goal is to make them as generic as possible so that all packages are able to implement the same set of interfaces irrespective of their unique implementation details. This entails not only deciding on the important components, but it also means determining the important data (and the data layout) that must be available to the different components. Information is passed between various components and classes to achieve the correct flow for the chemistry calculations. For example, the molecule should be set before its energy is calculated, which means the molecule object must be passed to the energy calculation model. This is also the point where classes and components are defined. The key distinguishing feature between class and component is that a class does not provide or use ports whereas a component does. So the developer needs to decide which functionalities are to be exposed to various components and which functionalities are needed only inside components. In our work, we have designed components that interact with classes and other components whereas classes interact only with wrapper functions (Figure 2).

Once the SIDL interfaces are written and the glue code generated through Babel, the next step is to implement the interfaces. In the case of NWChem, a wrapper function is used to act as a bridge between the CCA components and underlying computational package. This is not strictly necessary, but is useful in making the CCA interfaces simple and in writing high level routines in the native code that can be used for other purposes. Examples of wrappers include starting a package (e.g., setting up the package environment and the parallel tools, creating the run-time database), providing molecule coordinates, or getting the energy of a molecule.

As is true for much of software development, it is often true that the design needs to be revisited as the implementation is taking place since complicating factors often only surface during the implementation. So the design and the implementation often evolve together, as was the case for the QM/MM design. The final step is to provide and use component ports according to the data flow determined in the previous steps.

Figure 2. Ccaffeine framework and the underlying computational packages

For the QM/MM model, we initially need a QM model, which should be able to calculate molecular energy based on the molecule definition, theory level, and basis set provided by the user. The design of the QM model has been discussed in reference [11] and is shown in Figure 3. This was implemented using the same process as discussed above. However, the need to calculate the point charges from the environment was something that the original design did not take into account. Therefore, this has been added to the previous implementation. The interfaces are designed in such a way that this model can be used independently to calculate a purely quantum energy as well as in combination with MM to calculate the energy of the quantum region. A generic interface is the key to reusability. 

*Molecule* and *Molecule Factory* are used to set up the coordinates of a molecule. As discussed earlier, the QM molecule now needs to know about the external charges from the MM region and so these have been added to the Molecule and Molecule Factory interfaces. The QM Model and Model Factory are used to create an instance of a QM...
calculation and to associate the model with the molecule’s coordinates. Finally the driver component oversees the overall calculation to get the QM energy.

Figure 4. QM/MM component model

The next step is to have a MM component model which will calculate the MM energy based on the inputs given by the user such as the force field, cut-off distances and other constraints related to the classical region. The input for the molecule is read in through a Protein Data Bank [15] in our implementation, the MM input also identifies the MM and QM molecular zones. This is quite common in QM/MM implementations since the MM region is usually quite large compared to the QM region.

To enable the interaction between the MM and QM regions we introduce another component called Database to store and retrieve the intermediate quantum zone information and energies. We also introduce a QM/MM Model and Model Factory to handle the flow and interaction between the QM and MM regions. The QM component gets the quantum information from the database component inserted previously by MM component (after identifying the quantum zone in QM/MM).

We have meticulously developed a thin interface, which is required between the QM and MM regions in such a way that the moment the relevant information about the quantum zone is set by the MM component, the QM component uses the information to contribute its part towards the QM/MM energy. Defining the thin interface was a challenging task because final energy calculations are mostly dependent on the interface between QM and MM regions. The design layout is shown in Figure 4 (TAU Measurement component will be discussed later in section 6) and the sequencing diagram in Figure 5 depicts the sequence of flow and data between different components.

Figure 5. UML Sequence diagram for QM/MM calculation depicting the interactions between different components over time.
5. INTEROPERABILITY
Section 4 described the components developed for NWChem Package. We tested our design using the QM and MM model components developed with the NWChem code. The next obvious test was to check if we are able to use the QM/MM model with other packages like MPQC. It should be possible for other packages having quantum chemistry components to use the NWChem implemented molecular mechanics components for calculation of QM/MM energies (or gradients). Our proposed QM/MM model serves this purpose. As shown in Figure 6, we just need to load the QM components of the respective packages and other components of NWChem Package in the Ccaffeine framework. Our model flawlessly calculates the QM/MM energy by hiding the internal details, overcoming language barriers and input formats for different packages. Thus MPQC (and GAMESS) need not write its own piece of code for MM and QM/MM calculations, which would have otherwise called for large efforts by various scientists. In addition, theories that are unique to MPQC or GAMESS can be used to model the reactive part of the chemical system.

6. PERFORMANCE MEASUREMENT AND ANALYSIS
In this section, we discuss the overhead of using componentizing approach for the QM/MM calculation. Also comparison is done between the execution time needed to calculate QM/MM energy in a legacy code and time needed by using CCA components. To evaluate the execution time of the functions called in a legacy code as well as in CCA components, TAU [16] is used. TAU allows capturing of information at the node/context/thread levels. Using TAU, we perform source level code instrumentation by using interface calls at function entry and exit points. We have integrated the CCA components with TAU Performance component (Figure 4). Performance component provides MeasurementPort interface via which we are able to use the measurement capabilities provided by TAU in Ccaffeine framework. Both serial and parallel executions are possible using Ccaffeine framework.

<table>
<thead>
<tr>
<th>Molecule</th>
<th>Basis set</th>
<th># basis function</th>
<th>NWChm (seconds)</th>
<th>NWChem CCA Components (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tripeptide asa</td>
<td>sto-3g</td>
<td>14</td>
<td>0.93</td>
<td>1.085</td>
</tr>
<tr>
<td>tripeptide asa</td>
<td>aug-cc-pVTZ</td>
<td>210</td>
<td>325</td>
<td>374</td>
</tr>
<tr>
<td>ethane</td>
<td>sto-3g</td>
<td>16</td>
<td>1.36</td>
<td>1.571</td>
</tr>
<tr>
<td>ethane</td>
<td>aug-cc-pVTZ</td>
<td>260</td>
<td>1015</td>
<td>1122</td>
</tr>
</tbody>
</table>

Table 1. Execution time for calculating QM/MM energy

We have specifically used the timer capability provided by the MeasurementPort to calculate the execution time of a function. For testing, we chose two molecules: tripeptide alanine-serine-alanine and ethane. Table 1 shows the execution time and # basis functions for calculation of QM/MM energy using plain NWChem program. Also, it shows the execution time taken by component model of QM/MM for the same molecules and basis sets. These tests were performed on a SMP cluster of 4 nodes. Each node has a dual core Xeon 2 GHz CPU with 8GB of RAM and Red Hat Linux as the operating system. It can be seen that the overhead of componentization decreases as larger molecules are simulated. For sto-3g basis set, overhead is 16% and for aug-cc-pVTZ, it is 11%. Similar results can be observed.
while using MPQC’s QM factory and NWChem’s MM and QM/MM factories. Thus, it can be concluded from Figure 7 that the overhead reduces as the number of basis functions increases.

7. CONCLUSION

Componentizing a large-scale high performance legacy package is a challenging task. A variety of tools are used in combination such as CCA, BABEL, Caffiene framework, TAU, CCA-Chem Generic, MPICH, C++ and Fortran compilers that are often being updated. To provide consistency, it is required to recompile the packages and test them to ensure stability. The rebuilding process is time consuming and if errors or bug pops up, it is very difficult to locate the source of the error because of the various packages involved and to conclude which particular combination of packages may conflict.

In this paper, we outlined the process of creation of scientific components from legacy software using CCA specifications and discussed how to make these components truly interoperable with other components. We also discussed about the overhead of componentization and concluded that the overhead is negligible when compared to the overall calculation time. The main motive of this research work is to minimize the programming efforts of the scientists and at the same time not to impose a significant overhead as a result of componentization. The success of the proposed QM/MM component model will encourage further research and implementation of component engineering methods. This research is certainly very exciting as after the object oriented paradigm, component based software development in a high performance computational environment is the next big thing due to the numerous potentialities of component based applications. High performance component based development can skyrocket by demonstrating the benefits of component-based development to real life projects of significant size.

8. REFERENCES

[16] TAU http://www.cs.uoregon.edu/research/tau