Journal of Computer-Aided Materials Design (2006) 13:201–212
DOI 10.1007/s10820-006-9011-3

Software integration in multi-scale simulations: the PUPIL system

J. TORRAS, E. DEUMENS and S.B. TRICKEY*
Quantum Theory Project, Depts. of Chemistry and of Physics, University of Florida, Gainesville FL, 32611-8435 USA

Received 24 October 2005; Accepted 31 January 2006; Published online 25 July 2006

Abstract. The state of the art for computational tools in both computational chemistry and computational materials physics includes many algorithms and functionalities which are implemented again and again. Several projects aim to reduce, eliminate, or avoid this problem. Most such efforts seem to be focused within a particular specialty, either quantum chemistry or materials physics. Multi-scale simulations, by their very nature however, cannot respect that specialization. In simulation of fracture, for example, the energy gradients that drive the molecular dynamics (MD) come from a quantum mechanical treatment that most often derives from quantum chemistry. That "QM" region is linked to a surrounding "CM" region in which potentials yield the forces. The approach therefore *require*s the integration or at least inter-operation of quantum chemistry and materials physics algorithms. The same problem occurs in "QM/MM" simulations in computational biology. The challenge grows if pattern recognition or other analysis codes of some kind must be used as well. The most common mode of inter-operation is user intervention: codes are modified as needed and data files are managed "by hand" by the user (interactively and via shell scripts). User intervention is however inefficient by nature, difficult to transfer to the community, and prone to error. Some progress (e.g Sethna's work at Cornell [C.R. Myers et al., Mat. Res. Soc. Symp. Proc., 538(1999) 509, C.-S. Chen et al., Poster presented at the Material Research Society Meeting (2000)]) has been made on using Python scripts to achieve a more efficient level of interoperation. In this communication we present an alternative approach to merging current working packages without the necessity of major recoding and with only a relatively light wrapper interface. The scheme supports communication among the different components required for a given multi-scale calculation and access to the functionalities of those components for the potential user. A general main program allows the management of every package with a special communication protocol between their interfaces following the directives introduced by the user which are stored in an XML structured file. The initial prototype of the PUPIL (Program for User Packages Interfacing and Linking) system has been done using Java as a fast, easy prototyping object oriented (OO) language. In order to test it, we have applied this prototype to a previously studied problem, the fracture of a silica nanorod. We did so joining two different packages to do a QM/MD calculation. The results show the potential for this software system to do different kind of simulations and its simplicity of maintenance.

Keywords: multi-scale simulations, software inter-operation, QM/MM software

1. Introduction

The very nature of multi-scale simulation implies that algorithms investigated and implemented in this field involve the use of proven algorithms from many other fields. These component algorithms are themselves complex and sophisticated.

*To whom correspondence should be addressed, E-mail: trickey@qtp.ufl.edu

Often researchers in multi-scale simulations have resorted to reimplementation of the component algorithms. Another approach that has been used is to take existing implementations of the component algorithms and merge them into a single monolithic application capable of performing the multi-scale simulation [1–3]. Both these approaches are time consuming and error prone because they require significant changes to the codes that implement the component algorithms. As a result the multi-scale simulation researcher faces a large investment of time to study and revise unfamiliar codes and algorithms.

We propose the use of a general framework with standard interfaces to connect multiple applications in a way that requires only minimal changes to those components. We can then expect to reduce the time required to make component algorithms work together, allowing the multi-scale simulation researcher to spend the majority of his/her effort on developing and testing new algorithms addressing the multi-scale character of the problem.

There are, as already suggested, two different approaches to merging even two applications. The first way is to create a new application, e.g. by turning one application into a subroutine called by the other. This requires a careful study of both applications to ensure consistency of data and correctness of execution. A disadvantage is that when the maintainers of one of the original applications provide a new release, the analysis has to be repeated to again ensure consistency and correctness. The second way is to create a third application that runs the two original applications. If analyzed carefully this approach requires minimal changes to the applications. Typically a small subroutine to extract some data at some point in the execution suffices. Because the changes are minimal, a much smaller effort is required when a new version of the application is released.

The scientific problem providing the context for our work is the multi-scale simulation of crack formation in silica. The bulk of the material is treated with molecular dynamics (MD). However, the breaking of bonds is not sufficiently accurately described by the 2-body or 3-body forces available in the literature. Therefore we must describe a part of the system with quantum mechanical potential energy surfaces. A full ab-initio method is too costly, so a semi-empirical method is used. We choose the Neglect of Diatomic Differential Overlap (NDDO) method. The NDDO parameters were obtained by using the Transfer Hamiltonian approach [4]. The two applications that must be coupled then are molecular dynamics software, such as DL_POLY [5], to drive the dynamics, and semi-empirical electronic structure software, such as Thiel's MNDO [6], to compute the forces from the electronic potential energy surface. The task for our framework is to coordinate the applications during the dynamics and make sure that the forces in the chosen "quantum" region are the ones obtained from the semi-empirical software. In addition, there is a recently developed algorithm [7] to determine which atoms should be considered part of the "quantum" region and the framework must ensure proper execution of that algorithm as well.

2. Requirement analysis

A first view for the system structure is shown in Figure 1, where the prototype PUPIL (Program for User Packages Interfacing and Linking) system is what we ana-

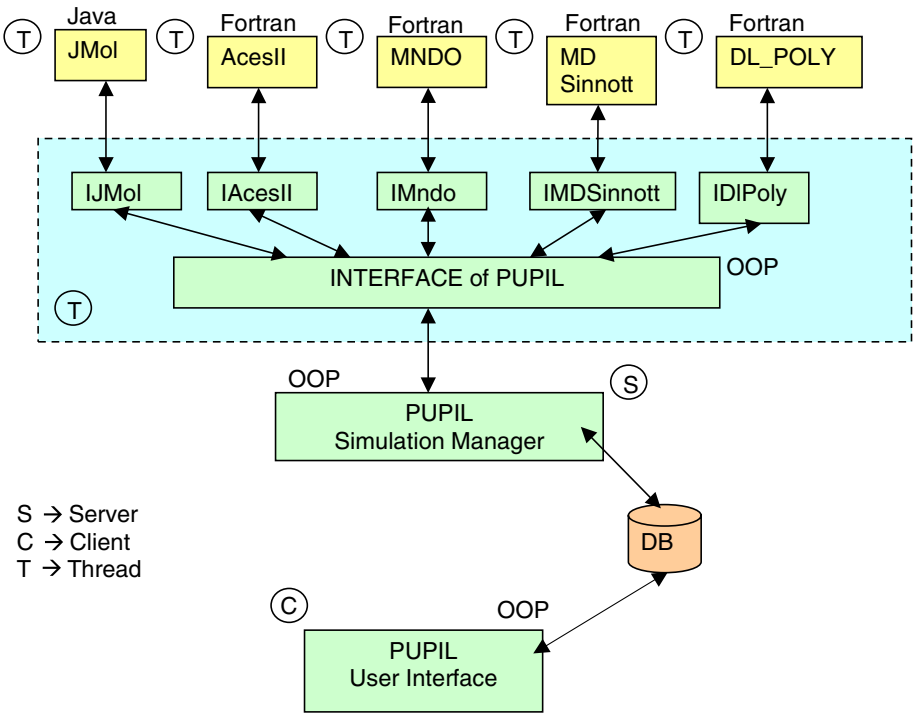

Figure 1. System requirements diagram to software integration in multi-scale simulation.

lyze in this work. We can see some package requirements to run a MD/QM simulation in two steps. The user interface allows preparation of the necessary data input. The simulation manager program interacts with different calculation packages asynchronously and exchanges information with the user at the end of the simulation. From this simple diagram we can expect three interfaces in the PUPIL system: User interface, simulation manager interface, and finally, a specialized interface for each application package or independent calculation unit.

The simulation manager coordinates all the independent packages following the instructions received from the user through a structured file. The Interface will connect different applications by exchanging information between them.

The external behavior of the software system is shown in two different diagrams (Figures 2 and 3): The first one describes the user interface and the second one describes the simulation server.

The user interface exhibits minimal functionality. It collects all the simulation information and the input files for the calculation components. All the information is packed in a structured XML file [8]. The same kind of structured file is used to display the simulation results obtained from the simulation manager interface.

The simulation manager needs to read the simulation XML file, start and manage the simulation, parse the output files, and finally store the results in the same XML format for the user interface program.

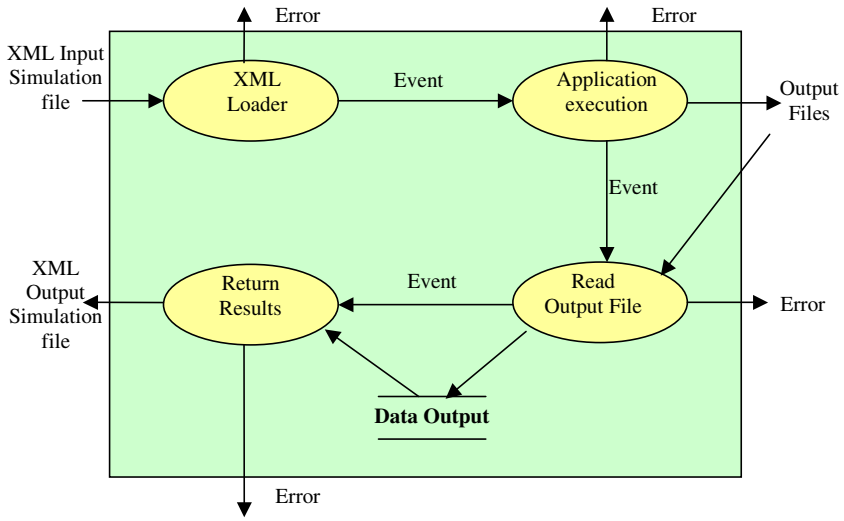

Figure 2. External behavior of User Interface.

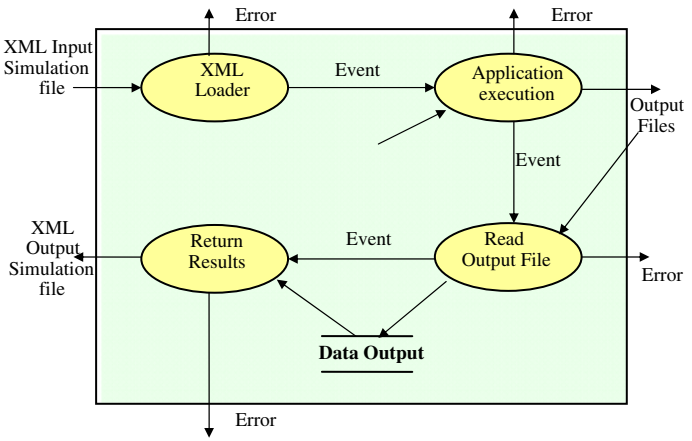

Figure 3. External behavior of Simulation Manager.

3. Specification

The main goal for the conceptual model is to show how the internal data is organized in memory (see Figure 4). The data structure can be organized into three memory regions: simulation data, input and output data files, and results. The simulation data is used to coordinate all the packages involved in the multi-step simulation. The second region is the Input and Output data file structures. All the files that have been parsed can be stored as a structured file using sections, variables, and values. The same structure can be used to store general input text files without parsing. Finally, the third region is the simulation result storage. This part must be specialized for the specific calculation units. Any future improvement must be done through building and using a general protocol of communication.

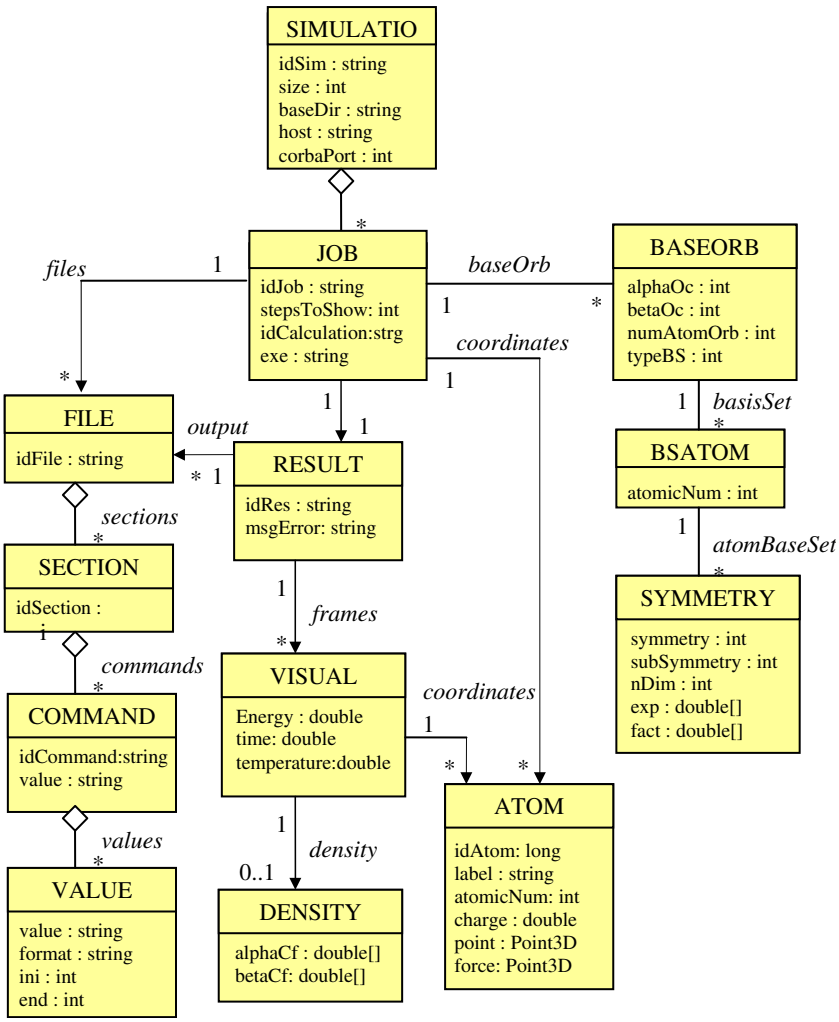

Figure 4. Conceptual Model.

The activity diagram indicates the basic synchronization between threads of server and application execution. The diagram starts with one thread and the bold lines show a fork process. The thread synchronizations are circles connected by a dashed line and the actions are shown in big ellipses.

The prototype specification has two external packages (MD and external QM force calculation) that must exchange information in the midst of the simulation. This coordination is to be done by the PUPIL simulation manager. To communicate between different programs we will use two CORBA [9] servers, which exchange position, force, and energy information at each MD step between the MD package and the simulation manager. In Figure 5 we observe the different threads for the simulation manager, MD and QM force calculation packages, as well as the CORBA server threads. The server threads are waiting to take the right action during the simulation when the client needs their services. Some extra synchronization points are needed to assure correctness and consistency in the general simulation execution.

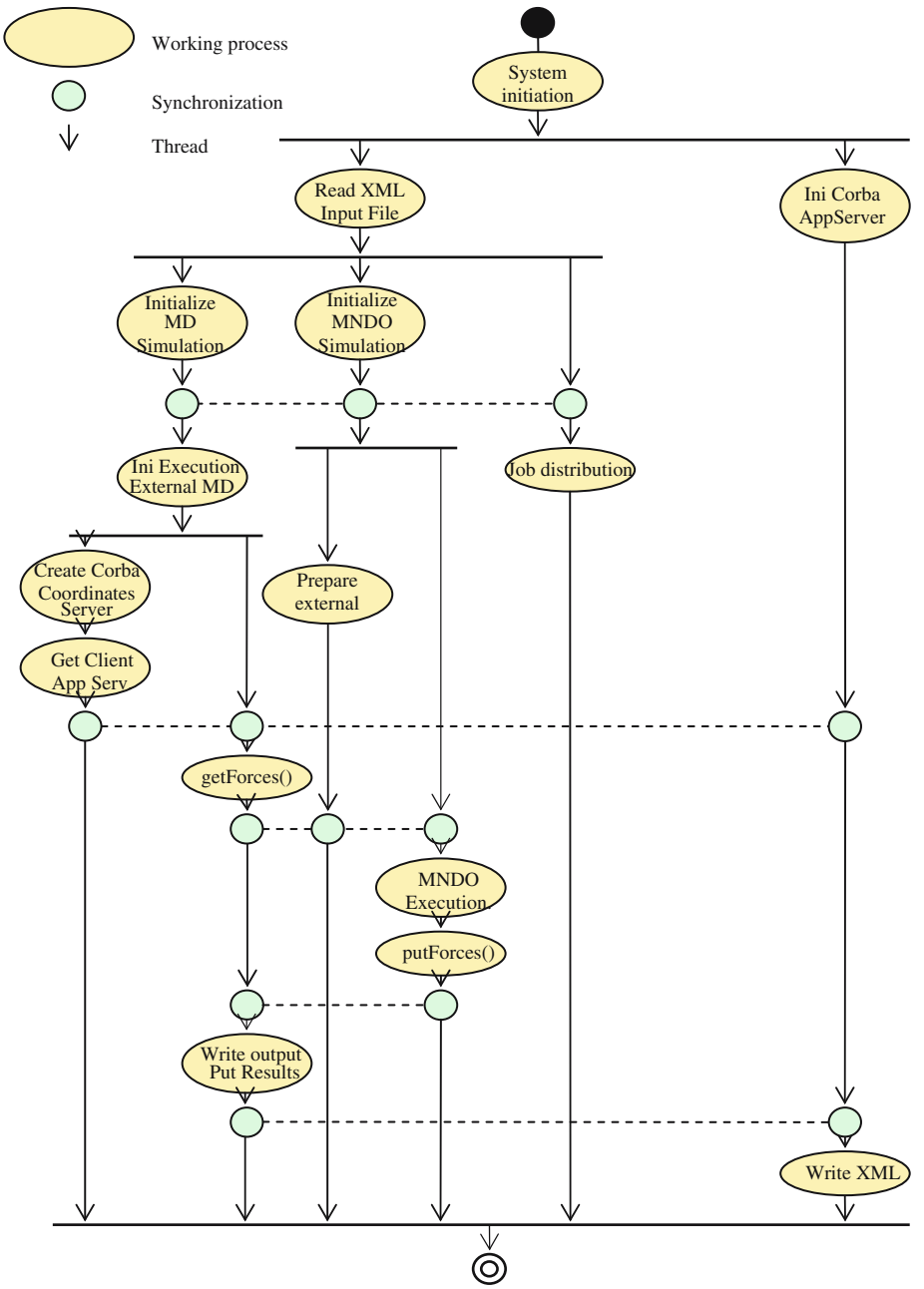

Figure 5. Activity diagram.

In this prototype, the QM package starts and the output information is being parsed at the end of the QM calculation for each MD step. That choice allows us to simplify the program architecture and avoid another CORBA server. The main goal for this prototype is to test this kind of architecture.

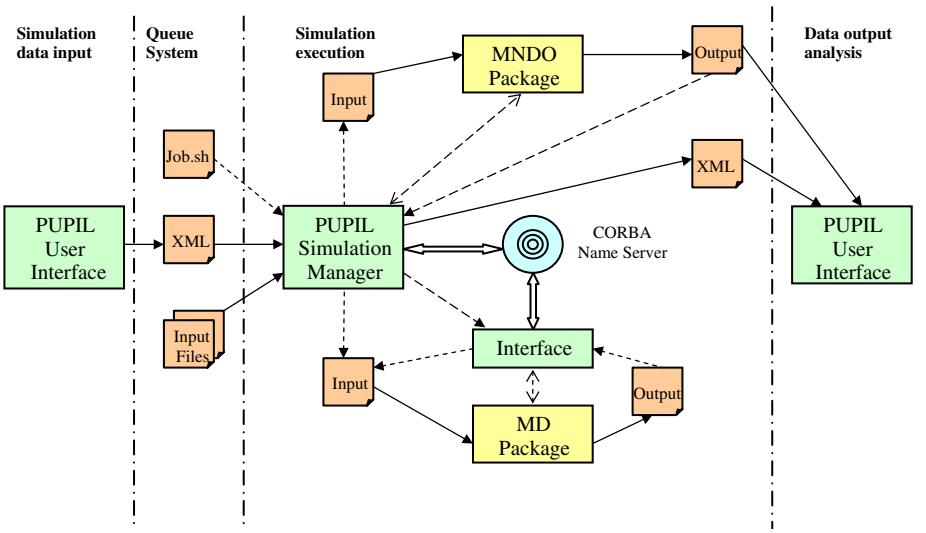

Figure 6. External behavior with time steps of simulation.

4. Design and implementation

As a result of the requirement analysis and specification, we have made certain design decisions which are determined by the implementation circumstances and the hardware facilities.

The job preparation and the simulation execution will be with asynchronous tools because the computer cluster that runs the simulation is controlled by a queue system; It decides when to run the simulation and which processor to use. That is the reason for use of a structured file to exchange information about simulation management, job preparation, and results between the PUPIL simulation manager and the user interface. At the end, the simulation manager will write a XML file with the results of the simulation that must be read by the user interface to extract the information properly.

Taking into account all the design and specification decisions, a new external behavior diagram can be drawn (Figure 6) for all the independent units integrated into the PUPIL system.

In this figure we show the four steps that make up the whole simulation process. In the first, the user builds a XML file with all the information required by the simulation. Next (second step), the queue system assigns the resources that will be needed. The third is the most complicated step and builds the core of the simulation. We can see in the picture how the main PUPIL server manages the simulation by starting the interfaces and calculation units. The communication between those units will be through CORBA space or through input and output data files. Finally, the user interface extracts the requisite information through the output XML simulation file.

The PUPIL user interface and simulation managers have been implemented in JAVA as a fast, and easy prototyping Object Oriented language. The resulting overhead will be shown later not to be relevant in the whole simulation. The use of JAVA gives us advantages such as multi-platform support, quick implementation,

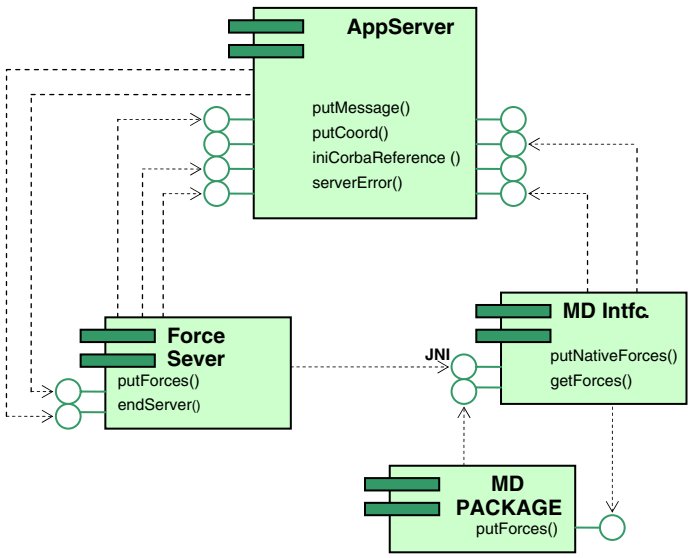

Figure 7. Components diagram.

easy maintenance, and software reuse. Only a small part of the simulation manager is platform-dependent: the wrappers between JAVA and FORTRAN. In the case of the prototype shown in this work, the specific MD interface uses two wrappers between JAVA and FORTRAN. The first interface has been built through the JNI (Java Native Interface) [10] as a bridge between JAVA and C, and the second allows interlanguage calls between the C and FORTRAN languages.

The diagram in Figure 7 shows how the components are connected and how the information flows. The MD/QM simulation starts with the MD, which queries the MD interface for the forces for a given set of system coordinates at every MD step. This request through the CORBA space reaches the simulation server. A MNDO calculation then starts and the force and energy are obtained through parsing the output file and are sent through the CORBA space to reach the force server. Following a JNI wrapper, the forces and energy reach the MD interface, which communicates the values to the FORTRAN code and finalizes a MD step. This process repeats until the end of the MD simulation.

The interface for the force calculation package writes the input files for the QM packages, starts the force calculation, and parses the output file to get proper information at each MD step. On the other hand, the interface for the MD starts the MD package and parses the output MD file once. This behavior could be extrapolated to every independent application package that must be plugged into the PUPIL software system.

All the components have been grouped by programming language. The simulation manager is composed of two files. The first one will be a Java Archive (JAR file) with the two CORBA servers and all the JAVA interfaces for the calculation units. The second one is a shared library, grouping all the machine-dependent software. On the other hand, the user interface code has one JAR file with all the components and a support library with the open-source JMOL molecular viewer project as a JavaBean.

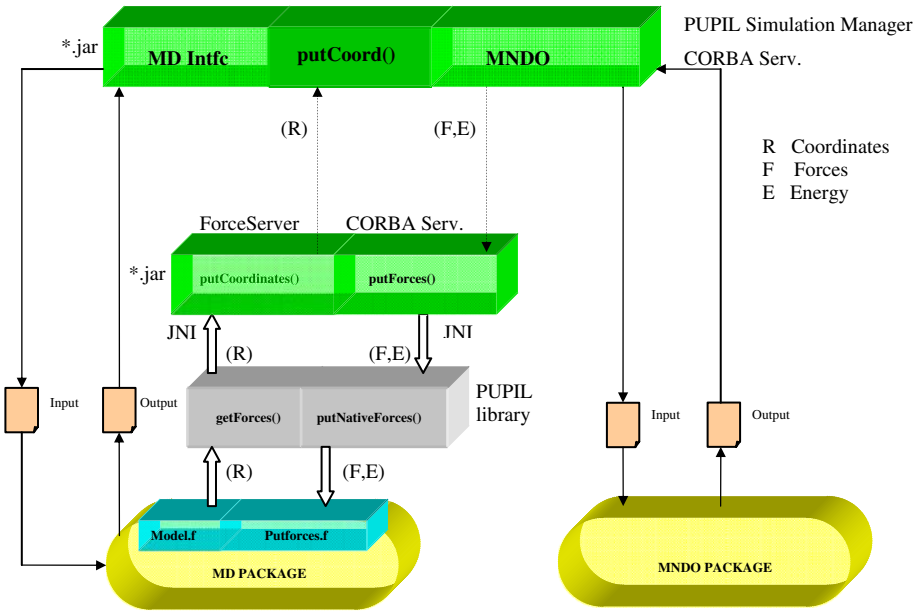

Figure 8. Components diagram from implementation point of view.

In order to simplify the prototype presented in this work we do not use a wrapper to interact with the MNDO code.

This stage of developments leads to a new components diagram with the implementation details shown in Figure 8. In this diagram we see the three components and their interaction through the function calls.

We have three main execution blocks. The first one is the main PUPIL simulation manager. The second one is the MD execution that connects to the main simulation server using a wrapper library and a JAVA CORBA client. The MD code needs two subroutines to translate the internal coordinate variables to the proper data structure required by the PUPIL package and the inverse function for force and energy values. The third block is the MNDO execution that is not modified in this prototype version.

The summary of file interaction for the prototype is shown in Figure 9. There are two application packages that must be plugged into the PUPIL system, the PUPIL simulation manager as a JAR file and its connection with the shared library. Another part is the user interface and the visualization java package called JMOL. This has been incorporated into the user interface as a Java Bean.

The user interface is not yet fully functional and has been implemented to support only the simulation. This software package could have two kinds of users; the first would be the end user who would use the functionality offered by the application packages already prepared to interact in a multi-scale simulation. The second kind would be a computational scientist who wants to plug a new application package into the PUPIL system.

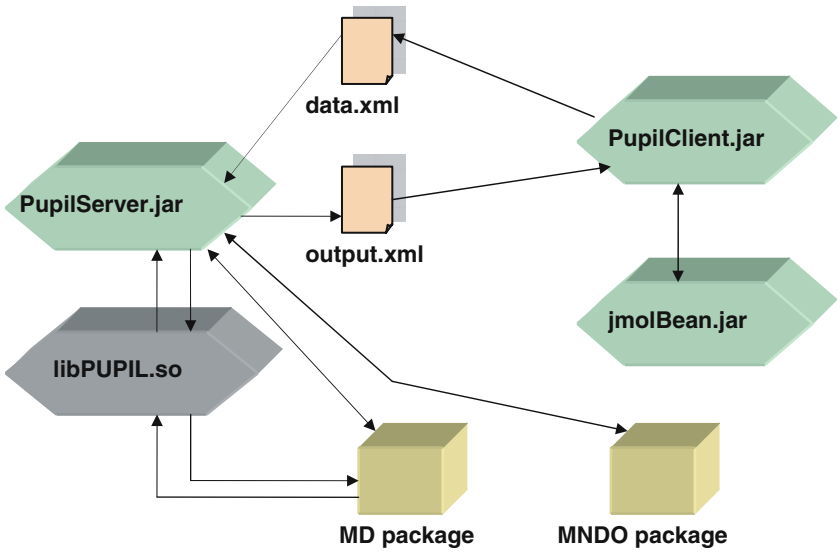

Figure 9. File Interaction.

5. Testing

The test aims to reproduce a previous calculation for a silica MD/QM fracture [11]. Uniaxial Strain has been applied to a silica nanorod molecule (Figure 10). This system has 108 atoms that are distributed as 6 silica rings with 12 atoms each one $(SiO)_6$ and 3 oxygen atoms to saturate the edge rings. The initial geometry has been obtained following previous work [11, 12] by a MD equilibration using the transfer Hamiltonian method and the TTAM and BKS interatomic potentials for SiO_2.

The strain applied over the nanorod molecule had been defined as a constant velocity of 25 m/s along the z coordinate for each atom situated on the edge rings. The MD code used here is different from the previous work, but both of them use a thermostat with velocity rescaling over 1 K that corrects the temperature almost at every step. The strain step is 0.001 Å every 2 fs and the external force calculation is done with the same QM package using the Transfer Hamiltonian (TH) approximation [4].

The final stress-strain comparison graph for the silica fracture of the nanorod molecule is shown in Figure 11. At the initial steps we can see a similar behavior for the two curves. That means a similar Young modulus for the elastic zone and a similar breaking point for the molecule.

We used a dedicated machine to determine the overhead associated with the communication between the servers through CORBA space and the writing and parsing of files for each MD step. The results obtained are shown in Table 1. In this table we can see that CORBA communication does not take too much time in comparison with the whole MD step. The overhead related to the I/O and communication takes 50% of the MD computational time, not taking into account the time associated with the external force calculation. This overhead could be significantly reduced if a wrapper interface were to be added to the MNDO package. The MNDO output file is a big file to parse and this takes a long time to process and write.

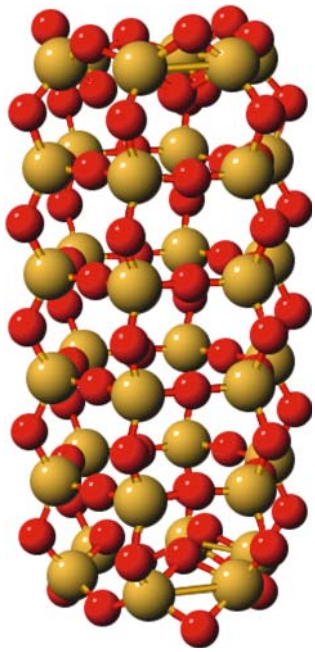

Figure 10. Nanorod System.

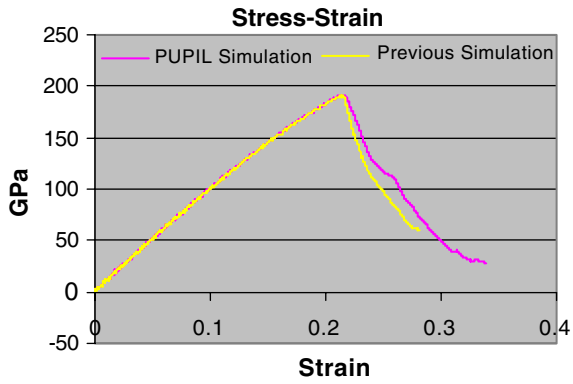

Figure 11. Stress-strain graph for the Nanorod system.

Table 1. Time process percentage for the main process involved into a Molecular Dynamic stress step in a dedicated machine

Time description	*%Real*	*%User*	*%Syst.*
Write input MD	0.20	0.02	0.08
Write input forces calculation	0.14	0.04	0.04
Putting coordinates (CORBA)	1.57	0.24	0.26
Forces calculation	79.53	78.89	76.95
Parsing forces output file	8.90	12.56	3.52
Putting forces (CORBA)	0.17	0.02	0.06
Parsing MD output files	0.09	0.10	0.04
Other PUPIL process	9.40	8.13	19.05
Time MD stress step	100.00	100.00	100.00

6. Conclusion

The design of the PUPIL package can be extended and improved in two ways. The first improvement will be to consider a quantum region and a classical region with selection controlled by the wavelet algorithm [7]. This will require keeping track of the history to apply the wavelet analysis and then obtaining the list of atoms to be treated with the quantum electronic potential energy surface from the wavelet analysis module. Secondly, the user interface could be improved in the direction of adding new user packages or introducing an expert system to guide the user into the multi-scale simulation.

The software architecture used for the PUPIL prototype shown here has been sufficient to coordinate the multi-scale simulation and merge different user packages into one simulation. The overhead in the communication between servers has been shown to be negligible. JAVA is sufficient to implement those servers. Small subroutines are needed to link any package to the PUPIL system and a considerable effort is necessary to write input files and parse output files once. The test did show the potential usability of this package on a real system.

Acknowledgement

Financial support from NSF ITR Grant DMR-0325553 is acknowledged.

References

1. Myers, C.R., Arwade, S.R., Iesulauro, E., Wawrzynek, P.A., Grigoriu, M., Ingraffea, A.R., Dawson, P.R., Miller, M.P. and Sethna, J.P., Mat. Res. Soc. Symp. Proc., 538 (1999) 509.
2. Chen, C.-S., Cretegny, T., Dolgert, A.J., Bailey, N., Myers, C.R., Sethna, J.P. and Ingraffea, A.R., Poster presented at the Material Research Society Meeting, (2000)
3. http://www.mcc.uiuc.edu/ohmms/
4. Taylor, C.E., Cory, M.G., Bartlett, R.J. and Thiel, W., Comput. Mater. Sci., 27 (2003) 204.
5. Smith, W. and Forester, T., J. Molec. Graphics, 14 (1996) 136.
6. Thiel, W., Program MNDO97 version 5.0.
7. Frantziskonis, G., Deymier, P.A., Phys. Rev., B 68 (2003) 024105.
8. Yergeau, F., Bray, T., Paoli, J., Sperberg-McQueen, C.M. and Maler, E., Extensible Markup Language (XML) 1.0 (Third Edition) W3C Recommendation 4th February 2004 http://www.w3.org/TR/REC-xml
9. Object Management Group, OMG Common Object Request Broker Architecture: Core Specification http://www.omg.org/
10. Sun Microsystems, Inc. Java Native Interface 5.0 specification
11. Taylor, C.E., Runge, K. and Bartlett, R.J., Mol. Phys., 103 (2005) 2019.
12. Zhu, T., Li, J., Yip, S., Bartlett, R.J., Trickey, S.B. and De Leeuw, N.H., Mol. Simulat., 29 (2003) 671.