Chapter 5

MeTA Studio: A Programmable Integrated Development Environment for Computational Chemists

5.1 Need for Programmable Environment

Chapters 2 through 4 elaborated on the CG-MTA algorithm and its use in computing one-electron properties, energy and gradient of a given geometrical configuration, its optimization and frequency evaluation. At the heart of all the CG-MTA calculations is the fragmentation procedure that cuts the parent molecule into a set of overlapping fragments, on which actual computations are performed to mimic the results of the supermolecule. The automated fragmentation procedure is powerful and general enough for most of the molecules. However, there is a frequent need to slightly alter the default fragments produced to get optimum speedup or accuracy. Moreover, the automated fragmentation procedure currently may not well represent the chemical intuition that a chemist may have and one would require a tool for assisting in manual fragmentation of a molecular system under study. Such a tool can be either based on a command line interface or a more intuitive graphical interface. Further, there is also a need for a visualization tool that can seamlessly handle files from a CG-MTA run including but not restricted to structure files, fragment files and scalar field files (such as MESP and MED). Although some available tools, for instance Univis[1], may address a few of these issues, one is left crippled with intricacies of a particular tool and has to always stick to a particular file format or a the way a particular task is done.

These restrictions of the available tools encouraged the author to venture into developing a new tool which does not have these restrictions. To a new user, it would provide all the tools that are necessary for easy handling of CG-MTA algorithms, assistance in manual fragmentation and visualization of calculation outputs from CG-MTA and other popular \textit{ab initio} packages. To an advanced user, it would provide and expose the complete underlying platform of libraries and provide a scripting interface to build in new features into the environment or change the way an existing interface works.
These ideas matured into an integrated development environment (IDE) and what now is presented in the ensuing Sections as MeTA Studio; a name which was originally derived from acronym MTA; but which is now general enough to support a broad range of other requirements of a computational chemist.

### 5.2 MeTA Studio: Extendable, programmable IDE tailored for Computational Chemists

Section 1.3 already made a strong case for the need of a powerful development environment for a computational chemist. Further, the previous Section emphasized on why such a tool is required for the current scenario of handling CG-MTA algorithms. To reiterate, tools such as PyMol[2] and Jmol[3] offer visualization and programming capability as was indicated earlier in Chapter 1. However, the programming capabilities of these tools are limited to exploiting the visualization capabilities offered by these packages. On the other hand, packages such as Mathematica[4] are starting to provide more and more tools to generalize scientific programming by providing auxiliary tools that could be used by non-mathematicians. In this case too, the programming interface is restricted to using the inbuilt language and to program with these environments using a language of your choice is not an easy task. Further, these technologies are proprietary and closed source. These are some of the major limitations that MeTA Studio aims to address. However, the aim here is completely different from the rest with the main focus being towards enhancing the productivity of a computational chemist, with special emphasis on handling large molecule. Moreover, the idea is to provide a complete IDE for a user to streamline his or her data analysis, collaboration and development needs.

MeTA Studio offers the following capabilities which makes it an aspiring platform of choice for developing future applications that enhance the productivity of computational chemists:

1. A programmable interface through BeanShell[5] as the primary language and support for Scheme as a secondary language. Other language support can be added as the need be.

2. Creating workspace, allowing to group various related files and tasks into a easily accessible workspace file.

3. Visualization and rendering support. Support for molecular visualization (along with scalar field visualization) is available using Java 2D graphics (default), JRMAn[6] library (rendering) and Java 3D (when available for online rendering).
(4) Rich set of MeTA Studio Application Programming Interfaces (APIs) and “Chem-Lets” for new application development in MeTA Studio or using it as a library. In addition, access to embedded Jmol[3] viewer and its libraries. The word “ChemLets” is used to describe the plugins and widgets that can be written in MeTA Studio or the once that are provided within.

(5) Access to a number of LGPL/ BSD [7] libraries including, but not limited to, quantumj[8] (QM calculation supporting HF and MP2), mathutils[9] (some common math routines including libraries for computing eigenvalues), jfreechart[10] (for charting and graphing), mp3 and ogg codec support and Google data API support.

(6) A peer-to-peer networking interface for grouping instances (called the Federation APIs) of MeTA Studio running on multiple machines to perform varied tasks including computation, rendering and real time collaboration.

Figure 5.1: Architectural diagram depicting main MeTA Studio components. The main MeTA Studio components are built over the standard Java platform and libraries which are also depicted in the diagram.

The architectural diagram depicting main components of the IDE are shown in Fig. 5.1. Following Section further elaborates over the design goals of MeTA Studio and its constituent components.
5.3 Design goals of MeTA Studio

MeTA Studio is designed with one prime goal: Programmability of each and every component in the IDE. The main focus of the IDE is towards structural computational chemistry. With this in mind, the components of MeTA Studio are built up using numerous design patterns[11] that make programming within MeTA Studio a simpler task. The major components in MeTA Studio are classified below and these form the building blocks of the features and services offered in the development environment.

(1) Subsystem for handling molecules, file formats and molecule related scalar fields.

(2) Mathematical subsystem providing basic classes for vector-, matrix- and some linear algebra operations.

(3) UI and graphics framework for displaying, manipulating and querying molecules, scalar fields and other auxiliary graphics.

(4) Workspace framework for organizing molecule, fragment files and scripts.

(5) Scripting framework for extending and embedding in MeTA Studio.

(6) Federation framework for peer-to-peer and collaborative computing.

(7) Fragmentation framework for handling CG-MTA or related Divide-and-Conquer methodology.

The IDE offers a seamless environment for a user that amalgamates visualization, collaborative computing, data management and a powerful scripting framework into a single platform. While basic tasks for visualization that support reading in of a number of file formats are supported, what makes the IDE’s visualization tool different from others is the possibility to write scripts to support reading a new file format that is not supported “out-of-box”. The same is valid for each and every framework that is provided in MeTA Studio.

In retrospect, the whole motivation behind developing MeTA Studio by the author is to offer a clean programmable and intuitive environment to work with CG-MTA algorithms and to provide a platform to explore other methods without worrying much about the underlying platform. During the development, however, the platform has itself matured to support a more general audience of computational chemist and is not restricted as a mere tool to support CG-MTA algorithms. To reduce the dependence on the underlying hardware and software platform, the Java programming language[12]
has been chosen as the platform for implementation of MeTA Studio code. The choice of Java platform not only provides for a clean and effective implementation of the code using Object Oriented programming paradigm [13] but also allowed the author to easily make available reproducible code on a different Java supported platform without recompilation.

In the following Section, we detail out some of the inbuilt features of the MeTA Studio and also examine the frameworks on which they are based.

### 5.4 Inbuilt features of MeTA Studio

MeTA Studio uses a simple philosophy for its user interface design: make commonly accessible tasks appear prominently and make others available as second level user interfaces or make them programmatically accessible. The common tasks that a user is presented with, changes dynamically depending upon the current user activity. This dynamically changing UI is implemented with in MeTA Studio as task panel and a “ribbon” UI (cf. Figures 5.2 and 5.3). For instance, when a user is trying to visualize a molecule object, the task panel and the corresponding “ribbon” interface shows the relevant options that are valid or related for the task being performed: such as saving the current geometry or saving the molecule image on to the disk. This task specific dynamic UI in MeTA Studio not only helps in reducing the clutter around the interface window but also provides a sort of non-intrusive help to a new user.

![Figure 5.2: Instance of dynamic UI in MeTA Studio. (A) and (B) displays the task panel in normal state and when the user is interacting with a molecular object respectively.](image)

All through, MeTA Studio features a similar task specific UI, which clearly streamlines the task(s) a user is trying to accomplish. MeTA Studio also incorporates common features for interactive help such as the status bar and the notification panel. These
Figure 5.3: Instance of dynamic UI in MeTA Studio. (A) and (B) indicate similar states (cf. Fig. 5.2) in the “ribbon” interface.

are used to provide a user with online help on the current task or provide blocking/non-blocking notifications for certain events or errors that may occur. These events may include change in state of a molecule being visualized, or completion of a job or a request from peer MeTA Studio user to initiate a talk session etc. A logging framework is also incorporated into MeTA Studio which logs all the user-initiated as well as system activities so that this information can be used for tracing bugs or problems in MeTA Studio. The logging information is also useful for debugging scripts that can be written inside the IDE.

Visualising molecules, scalar fields and workspaces

Visualizing molecular structures from standard file formats such as XYZ or PDB is one of the most common tasks that a computational chemist routinely does. MeTA Studio provides a streamlined viewer (and associated framework) for visualizing, querying and rendering molecule objects and its related scalar field properties. In case, a user wants to utilize a different viewer instead of the default, for instance the JMol viewer, this can be easily achieved by writing appropriate scripts. The current implementation already includes wrappers for using the JMol viewer instead of the default IDE viewer. Additionally a user can write scripts to parse and display a file format that is not directly supported by the IDE, as will be exemplified later in Section 5.5.

The default molecule viewer provided with the IDE incorporates most of the common visualization tools provided by other molecule viewer. These include support for various molecular models such as line model, ball-and-stick model etc. The viewer also incorporates features to allow visualization of various scalar fields (such as MESP and MED) as contours, isosurfaces, textured vdW surfaces or fuzzy volumes, which are conceptually similar to those supported in Univis[1]. However, the viewer differentiates itself from the rest in providing some innovative ways of querying the molecule object and related properties using a powerful but simple, one-line, comma separated query
Figure 5.4: *Find* tool of the inbuilt IDE viewer in action. (A) Interface description of the *Find* tool that appears at the top of each window showing the molecule object. (B) The *Find* tool can be used to query structural properties such as length, angle and dihedral. In the figure a zoomed image of the queried structural property is shown. (C) All the queries and their results are logged into the notification panel that can be accessed at a later time.
language. The query language is especially useful for searching a particular parameter for a large molecular structure, as well as introspecting for certain values over a large grid representing a scalar property. The viewer incorporates a tiny “Find” tool (cf. Fig. 5.4 A) that provides a simple UI to enter, evaluate and get immediate feedback on results. If querying for a structural property such as bond angle, the molecule is appropriately translated and rotated so as to bring the relevant atoms and the result of the query to the center of the screen (cf. Fig. 5.4).

The Find tool can also be used to query a grid based scalar property attached to the currently displayed molecule. In summary, the Find tool provides a powerful query language to easily navigate molecular and related scalar field data and can be extended in future to include other format specific utilities such as querying a PDB file for presence of a particular amino-acid.

Molecules and one-electron properties that can be visualized in the IDE viewer can be saved as image files or rendered on-line using Java3D renderer or off-line via a RenderMan[14] compliant renderer such as JRMan[6]. Other renderers such as PovRay[15] are not supported at the moment. However, this support can be externally added via writing scripts that use the rich set of APIs provided with MeTA Studio. All the graphics used in this thesis is rendered with in MeTA Studio using JRMan library.

Figure 5.5: The multi-view feature in the MeTA Studio viewer allows one to have multiple camera views of the same molecule scene. The screen shows water molecule with zero valued MESP contour and its various camera views.

Another feature that differentiates the inbuilt viewer from other viewers is the support for simultaneous multi camera view of the same molecule scene (consisting of the molecule as well as the attached properties). For instance Fig. 5.5 depicts a water molecule with zero valued MESP contour and its various camera views. A user can conveniently switch between any of the views to access the required information. Views can also be dynamically created and destroyed as needed by the user. Molecule level visual properties such as atom selection and highlighting are synchronized across all
visible views. This feature is especially useful when performing manual fragmentation of 3-D molecules, where in multi camera views of the molecule easily aid a user in selecting appropriate set of atoms as a fragment.

A computational chemist working on a particular problem would mostly keep all the related files such as structure files, outputs of calculations, scalar field files arranged together for later analysis. This is usually done by utilizing the underlying file system. However, the underlying file system, in general does not understand the formats of the files used by a computational chemist, which, many a time, adds to confusion and loss in productivity. To aid in arranging and managing various files related to the task at hand, MeTA Studio provides a workspace based framework and related UI. The workspace framework in MeTA Studio provides an additional abstraction over files and exposes them as objects relating to the work, such as a molecule object, a property object or a script object. This abstraction aids a user to better organize the task at hand as related objects can be grouped together, such as a molecule object might have various fragmentation schemes attached to it etc. Using a workspace also saves the time required for routine parsing of output or input files for visualization as the workspaces are optimized with a lot of meta-data added to them and bypasses expensive operations such as forming connectivity matrix from molecular coordinates.

**Manual fragmentation**

One of the motivating factors behind building MeTA Studio was for building a tool that can assist in construction of different fragmentation schemes (manual or automated) and allow for comparison of these. Fig. 5.6 depicts a set of cutting tools provided in the IDE that can be used to graphically cut a molecular system into fragments.

![Various cutting tools provided in MeTA Studio. A tool for individual selection of atoms, another for a pencil based arbitrary selection of atoms and two geometry based (cube and sphere) cutting tools are provided.](image)

Though the manual fragmentation procedure can be used with any molecule object, it is best to use a workspace so that the related fragment objects can be coupled together with the molecule object. To illustrate how a manual fragmentation routine
works, we take a small test case of α-tocopherol and run through the screens of what can be done with this tool (cf. Fig. 5.7).

Figure 5.7 shows main interface involved in the manual fragmentation tool. A user first encircles the atoms that he/she would like to be included in a fragment. After the selection process is over, a list of options indicating what can be done with the selected atoms is presented (cf. Fig. 5.7 A). The simplest option adds the new fragment to the workspace with default options, or else a user can choose to modify some parts of the fragment and then add it. Once the fragment is added to the workspace, the fragment is visually highlighted in the viewer along with the parent molecule (cf. Fig. 5.7 B). A goodness panel embedded in the interface also keeps track of the atomic goodesses as the user adds more fragments to the workspace (cf. Fig. 5.7 C). The goodness panel can be used to find out the atoms having least $R$-goodness and these can be manually corrected for by centering a new fragment on the problematic atom. Once all the main fragments are generated, one can also get the cardinality fragments (or overlap fragments) by choosing appropriate tool in workspace interface (cf. Fig. 5.7 D). The fragments thus generated can now be exported in a form that can be read in by CG-MTA code (cf. Section 3.2) or exported in many other supported formats such as XYZ and Gaussian input files. The workspace also provides a utility to add dummy atoms to the fragments that are manually generated. This is particularly useful if the user intends to run the fragments using a different \textit{ab initio} code other than CG-MTA modified GAMESS. Finally, if fragment energies are computed externally, they can be read into the workspace and total electronic energy of the parent system can be evaluated based on the cardinality expression (cf. Section 2.6).

Thus, the manual fragmentation utility serves as a powerful tool and aids in easy access to many tasks offered by CG-MTA code described in previous Chapters.

\textbf{Multicore support}

Chapter 1 had mentioned of increasing trend and popularity of multi-core processors. However, most of the visualization tools developed or available do not in general take cognizance of the presence of more than one core. This may seriously undermine the scaling of such tools on next generation processors which increasingly offer more cores per processors package. MeTA Studio, however, follows a different path and many of the compute-intensive parts of the IDE code are written in a fashion to take advantage of all the available cores. Some of the places where multi-core enabled code is used include: formation of connectivity matrix, identification of weak bonds, calculation of
Figure 5.7: The manual fragmentation interface in MeTA Studio. (A) The selection tool presents with options to make a new fragment. (B) Once the fragment is made, the fragment portion is highlighted and added to the workspace object tree. (C) An automatically updated goodness panel shows the current goodness of each atom. (D) Additional utilities for handling CG-MTA algorithm.
one- and two-electron integrals etc. The parallelization in MeTA Studio code is currently achieved by simple outer loop splitting procedure, with the sub loops executed using different threads, one per each core.

**Tracking CG-MTA Jobs**

Section 3.4 has indicated that a CG-MTA job can send notifications on its current status to a pre-specified server. This information on the current job status is sent as XML[16] file to the notification sever. MeTA Studio can act as a client to this notification server and can periodically fetch the information and display useful information on a currently running CG-MTA job. Information such as the current and previous energy, gradients and current geometry are particularly useful for monitoring a long running optimization job. Fig. 5.8 depicts one such scenario where an optimization job for a molecule (code named bisfc) is shown. Note the details such as energy and gradients for the optimization run along with options to save or view the current geometry are displayed; another nifty feature in the IDE designed to streamline a user activity.

![Image](image.png)

Figure 5.8: A simple but informative Job monitor UI for remotely monitoring a CG-MTA job is incorporated into the IDE.

**5.5 Programming in MeTA Studio**

As indicated earlier, each and every subsystem of MeTA Studio has been developed to be programmable right from the beginning. Programmability in MeTA Studio can be achieved in two major ways: first using the inbuilt scripting framework (the default scripting language being Java like BeanShell scripting language, but other lan-
guage support can be easily added), the second is to use MeTA Studio APIs as a library to build completely new applications.

MeTA Studio APIs heavily uses numerous design patterns[11] and most commonly followed object oriented programming practices[13] so as to make programmability painless and more straightforward. The MeTA Studio APIs are arranged into Java packages and classified as separate subsystems. Each subsystem is independent of other and interacts with the other using defined set of constructs. Most top level, routinely used objects such as Molecule object, are not concrete classes, but are rather interfaces or abstract classes with the implementation of these hidden away from user level scripts. The main set of packages and their purpose is enumerated below. A detailed description of these can be found in on-line help incorporated in the MeTA Studio package[17].

(1) **org.meta.molecule** and **org.meta.fragment** packages provide various classes related to handling molecule and fragment objects. Along with these other two packages, **org.meta.propertyreader** and **org.meta.moleculereader** offer classes to represent a grid based scalar property and a framework for reading in various file formats (for example XYZ, PDB or outputs from GAMESS or Gaussian programs) respectively. The framework for reading molecule files is also written to make it accessible from scripts so that a user can enable MeTA Studio to allow to read in his or her own file formats.

(2) **org.meta.math** and **org.meta.common** packages offer common mathematical and other utilities. This package provides classes for handling Vector and Matrix objects as well as framework for matrix diagonalization and related matrix operations.

(3) **org.meta.scripting** provides a simple scripting framework for supporting multiple scripting languages in MeTA Studio.

(4) **org.meta.net** incorporates classes for networking and federation framework in MeTA Studio. These classes provide a framework for building collaborative tools to interact among peer MeTA Studio users. Sample implementations, including a talk facility, are incorporated into this package. The talk facility included in this package demonstrates some of the unconventional things that can be achieved in a collaborative environment. For instance, the talk session instead of being a pure text based interface, incorporates object interactions. It means that users can share molecule or other objects (such as scripts) embedded
in the chat session. This essentially creates an intuitive platform for interactive collaboration among computational chemist working on a single large problem.

(5) \texttt{org.metaworkspace} is the package that provides a framework for handling workspaces with in MeTA Studio. This package also provides all the basic classes necessary for creating and modifying a workspace in the IDE.

(6) \texttt{org.meta.shell} provides all the classes pertaining to the GUI of the IDE. This package is basically an amalgamation point of all the above listed packages into a single user presentable interface. All the graphics related and UI related classes (including the inbuilt molecule viewer) are a part of this package. All the classes in this package are designed in a way in which they can still be used to construct new application from “reusing” the UI code as an external library; one such example will be discussed in Section 5.6.

MeTA Studio uses design patterns\cite{11} such as singleton, observer, adapter, factory and a combination of these patterns for developing the above enumerated list of packages. A user aware of these patterns can invariably take advantage and would be easily able to use the set of APIs that the IDE provides. Even when one has no clue of what design patterns mean, they allow for a logical description of APIs provided, making it easier for getting the job done!

MeTA Studio’s scripting framework provides two additional sub-frameworks: (i) To write extension scripts in MeTA Studio and (ii) To build widgets that expose new UIs with in MeTA Studio. The extension scripts, also termed as plugins are meant to introduce additional BeanShell functions to offer new functionalities not already supported in the IDE. Widgets, on the other hand, are designed to provide more than simple functional extensions. Rather they are designed to be integrated into the IDE interface to offer completely new set of features and the related UI. Both plugins and widgets are automatically loaded when the IDE boots up. Plugins are however more dynamic and they can be created “on-the-fly” as and when the IDE instance is running, on the contrary a widget is loaded only once after the IDE has booted up. To load a new widget one would require to restart the IDE instance. The following Section will provide some working examples of plugins and widgets that demonstrate the ability to extend functionalities of MeTA Studio.
5.6 Examples of extending and embedding in MeTA Studio

Previous Sections have given a brief introduction to the powerful programmable environment provided by MeTA Studio. In this Section, a few real usage scenario of MeTA Studio APIs are presented.

Earlier on in Section 2.10, WebProp, an online system for evaluation of *ab-initio* quality one-electron properties was introduced (cf. Section 2.10). One of the requirements while building WebProp interface is to provide a user with an online viewer that also allows for interactive construction of user defined grid on which to evaluate a chosen scalar field (such as MESP or MED). The first and foremost natural choice was to use MeTA Studio viewer for this purpose. As MeTA Studio is written in the form of a desktop application rather than an applet, it was decided to use MeTA Studio APIs as an external library, so that it can be supported on a Java enabled Web browser. The IDE’s user interface code being designed in a manner which promoted easy re-usability, writing an adapter class to expose the MeTA Studio inbuilt viewer as an applet did not turn out to be a very cumbersome task at all. Fig. 2.11 in Chapter 2 depicts the resultant UI. With in the code, the classes in the packages `org.meta.shell.idebeans.viewers` and `org.meta.shell.idebeans.viewers.impl.moleculeviewer` were extensively used to construct the UI. Fig. 5.9 depicts the top level class interaction diagram for the applet UI in WebProp.

The adapter class (*MeTAppletAdapter*) is written in Java and is composed of instances of objects `MoleculeViewer` and `Molecule` classes. Note that `MoleculeViewer` object is written in a fashion that can be used easily outside the IDE, and can be programmed externally. The adapter class exposes some of these features such as ability to change the display model, or the grid size encompassing the molecule to the relevant HTML controls via Python scripts. The `Molecule` class is an abstract class, and whose
implementation may change at runtime depending upon how the MeTA Studio library is configured. The instantiation of this class is not done using standard object creation techniques, but rather uses Java's reflection APIs[12]. In MeTA Studio, a number of such classes where the concrete implementation is hidden from user level scripts, employ reflection APIs for creating an object instance. An interesting observation in the above example is that the code required for the applet adapter used in WebProp contains not more than 100 lines of Java code! Thus building new applications related to structure based computational using MeTA Studio APIs can heavily reduce the amount of code that might be required.

MeTA Studio also provides an alternative and more intuitive way of programming: via purely scripting framework. To give a simple “Hello, World!” example using the default scripting interface, i.e. BeanShell refer to the following piece of code:

```java
hello(name) {
    print("Hello, " + name);
}

hello("MeTA Studio!");
```

One can easily observe that the BeanShell syntax is quite similar to the one offered by Java programming language, sans the strict object oriented syntax enforced in the later. A more detailed description of BeanShell and the related changes introduced in MeTA Studio can be found elsewhere[5, 17]. BeanShell, apart form providing access to the MeTA Studio APIs, also provides full access to the Java Standard libraries allowing a user to develop more general applications within MeTA Studio. BeanShell as incorporated into MeTA Studio provides several additional commands for easily building new applications or just getting some quick work done. The following piece of code shows how easy it is to programmatically create a molecule object and then show the structure in the inbuilt IDE viewer.

```java
// create a molecule object and call it water
mol = molecule("water");

// add atoms to the molecule object
mol.addAtom(atom("H", 1.0, point(0.752510, -0.454585, 0.000000)));
mol.addAtom(atom("O", 6.0, point(0.000000, 0.113671, 0.000000)));
mol.addAtom(atom("H", 1.0, point(-0.752510, -0.454786, 0.000000)));

// build connectivity
buildConnectivity(mol);
```
A number of such functions have been used to build an alternative interface to the WebProp system which is incorporated with in MeTA Studio as a widget. The purpose of the widget is to make WebProp job submission even easier by integrating it within MeTA Studio using the widgets interface. Fig. 5.10 shows the main steps involved in submitting a job via the WebProp widget. The WebProp widget presents a simple three step UI, wherein the user first selects a molecule for which he is interested in computing a property (scalar field). This molecule is shown in the same molecule viewer described in earlier Section, the difference being that, in this case, it is incorporated in a different setup altogether. Once the required molecule is selected, the user is presented with a list of options (i.e. the kind of computations to be performed, basis set etc.). The final step involves indicating the job identifier for the job to be submitted from the WebProp server, as well as provides a text box to key in the e-mail address to which the user would like the results to be sent. In the backend this widget “talks” to the WebProp server and transmits appropriate information to initiate a job submission. In case any error occurs during the submission process, appropriate error message is flashed.

In a nutshell, MeTA Studio’s powerful programmable environment provides programmatic access to all the components it is made up of and in the process aids in rapid development of applications related to computational chemistry which otherwise would take up lot of time and resources.

5.7 Note on Federation framework

An additional feature in which MeTA Studio differs from other existing frameworks and visualization tools is the support for networking (also called here as federating) a number of peer MeTA Studio instances. Each instance of MeTA Studio can publish a service on the network, examples include a talk service or a simple service to receive text messages, or yet another service to allow remote access to output files of certain running job. The federation framework allows for automatic discovery of MeTA
Figure 5.10: The WebProp widget embedded in MeTA Studio. (A) General WebProp widget and molecule selection UI. (B) Select the basis set and type of calculations to be performed. (C) The final step indicates the job identifier as well as provides a text box to enter in the user e-mail address to which the results are to be mailed.
Figure 5.11: A talk session between two users with in MeTA Studio. Users can exchange objects such as molecule and scripts apart from usual text, providing an intuitive environment for collaborative work.
Studio instances as well as services offered by these. A user can interact with the federation framework in a number of ways. One way is to pool the compute resources offered by each of the MeTA Studio instances and form an *ad-hoc* cluster that can be used to solve a particular problem (such as performing SCF on fragments or a set of molecules). Other instance wherein the federation framework is useful is for active collaboration among various computational chemists. For instance, MeTA Stdio provides a `talk(<ip address>)` function and a small talk widget that allows two or more users sitting on different terminal to interact and share objects such as molecules and scripts. In effect, this provides for an intuitive environment for collaborating on a single problem in real time even if separated by geographical boundaries. Fig. 5.11 depicts and typical situation where in two MeTA Studio users are sharing information over a talk session. Note the ease with which several objects such as molecule and scripts can be shared among users.

In a typical collaborative environment, it but absolutely necessary to account for the security aspects of data exchanged between the peers. The talk interface and the federation framework, on which it is based, follow a simple rule based security policy so as to control how different MeTA Studio instances can communicate with each other. Users can manually block access to a particular service for a particular IP address. The default behavior of the rule based policy as implemented in MeTA Studio is to prompt user on each new request (such as a talk request) made from a peer user. Further, all the network traffic between instances of MeTA Studio is carried out over a secure socket layer (SSL) instead of commonly used plain sockets.

In conclusion, the techniques and tools mentioned in this Chapter have been extensively developed for usability and programmability that is especially tailored for a computational chemist. While inbuilt tools are provided for visualization and collaboration, the programmability of the IDE itself opens up new vistas of possibilities of user programmed functionalities. The complete code as well as the binaries are available[17] under the LGPL license. A few features mentioned in the current Chapter are at a primary stage of usability, but in future the author plans to incorporate several new features to have streamlined functionality even across different instances of MeTA Studio.

**References**


[7] GNU, GPL and LGPL licenses are offered as copy left licenses for distribution of free and open source software. While GPL (GNU's General Public License) mandates one to release all source and derived source, Lesser-GPL (or LGPL) is basically meant for developing libraries or APIs wherein a derived piece of software may not be disclosed. BSD (or Berkley Software Distribution) was a license originally used to distribute a flavor of UNIX developed at Berkley University, but now is used as an alternative opensource license to distribute many codes., http://www.opensource.org/, 2007.


[16] XML, eXtendable Markup Language is a way to arrange information (usually textual) using tags or markups in such a way that it is format independent and can be parsed by a standard XML parser to reliably retrieve back the information., http://www.xml.org/, 2007.