

# JACOB: An Enterprise Framework for Computational Chemistry

Mark P. Waller,\* Thomas Dresselhaus, and Jack Yang

Here, we present just a collection of beans (JACOB): an integrated batch-based framework designed for the rapid development of computational chemistry applications. The framework expedites developer productivity by handling the generic infrastructure tier, and can be easily extended by user-specific scientific code. Paradigms from enterprise software engineering were rigorously applied to create a scalable, testable, secure, and robust framework. A centralized web application is used to configure and control the operation of the framework. The application-

programming interface provides a set of generic tools for processing large-scale noninteractive jobs (e.g., systematic studies), or for coordinating systems integration (e.g., complex workflows). The code for the JACOB framework is open sourced and is available at: [www.wallerlab.org/jacob](http://www.wallerlab.org/jacob). © 2013 Wiley Periodicals, Inc.

DOI: 10.1002/jcc.23272

## Introduction

Modern scientific computing has advanced over recent years, with computational chemists having access to immense computing facilities.<sup>[1]</sup> There are two common strategies for addressing scalability issues in computing: vertical scaling which increases the resources (and hence performance) for a given node, or horizontal scaling where additional nodes are added to the system to increase the systems overall performance. Although vertical scaling obeys the well-known Moore's law,<sup>[2]</sup> horizontal scaling appears to be directing the future of scientific computing. Distributed computing, which couples a cluster of computers coupled via a network, comes with its own set of challenges: for example, complicated programming models are required to solve issues such as throughput (volume of data transferred across the network) and latency (the time taken for a packet of data to be transferred between two points) that occur with horizontal scaling. Academic projects have been ported to a number of distributed computing environments such as grid computing<sup>[3–5]</sup> open science grid (OSG),<sup>[6]</sup> volunteer computing<sup>[7,8]</sup> Berkeley Open Infrastructure for Network Computing (BOINC),<sup>[9]</sup> infrastructure as a service,<sup>[10]</sup> and, most recently, to cloud computing. Scientific computing in these environments poses significant challenges when increasing the size of datasets (scale) or when coupling between different software components (systems integration). To meet these challenges, scientists have created generic workflow packages. A workflow can be used as a general tool to orchestrate a complex task into a series of more simple reusable tasks. The workflow can also be used to manage repetitive tasks that are required for processing large-scale datasets. These general-purpose scientific workflows handle the

generic infrastructure requirements and typically offer a wide range of common prewritten tasks. Importantly, these workflow solutions are extensible, meaning that users are able to extend the functionality of the workflow when required for instance by creating a new task. Generic workflow packages such as Kepler,<sup>[11]</sup> Knime,<sup>[12,13]</sup> and Taverna,<sup>[14]</sup> whereas Chemshell<sup>[15]</sup> (Tcl<sup>[16]</sup>) and the more recent PyADF<sup>[17]</sup> (Python<sup>[18]</sup>) are more suited to systems integration in the area of quantum chemistry. The WebMo<sup>[19]</sup> web server couples a user-friendly web flow (workflow concept adapted to a web context) to a range of computational chemistry packages via interfaces, whereas Gabedit<sup>[20]</sup> is a more recent graphical user interface for computational chemistry. Overall, all these solutions are able to dramatically increase the productivity of a scientist once they are familiar with a particular software package.

Enterprise is a term that describes organizations such as corporations and government institutions. Such organizations have special requirements for software regarding scalability, robustness, and security. The robustness requirement comes from the critical nature of these applications, where downtime, for example, may cause significant revenue loss in the financial sector. Security is also a concern based on the sensitive nature of financial transactions, or the data protection needed for government institutions. Therefore, software engineering professionals have met these requirements by constructing so-called "enterprise software" solutions. These applications tend to be data-centric, and design patterns have emerged as being particularly useful guidelines. In enterprise environments, the requirements for integration between uncoupled software

---

M. P. Waller, T. Dresselhaus, J. Yang  
Theoretical Chemistry, Organic Chemistry Institute, Westfälische Wilhelms  
Universität Münster, Correnstrasse 40, Münster 48149, Germany  
E-mail: [m.waller@uni-muenster.de](mailto:m.waller@uni-muenster.de)

Contract/grant sponsor: Deutsche Forschungsgemeinschaft; Contract/grant number: SFB858.

© 2013 Wiley Periodicals, Inc.

---

\*The lesser known Wirth's law that states 'software is getting slower more rapidly than hardware becomes faster.'

components across multiple networks and repositories present a significant set of challenges. Enterprise software engineers have developed central administration interfaces and rely heavily on the messaging design pattern to meet these challenges.<sup>[21]</sup> Another feature of enterprise software is their reliance on frameworks. A framework is an abstraction where the software provides generic functionality that can be changed by user code, thus providing application specific software. A framework is therefore a universal and reusable software platform for the efficient development of applications. Different framework infrastructures are available for enterprise software development. Recently, frameworks that use the “inversion of control” (IoC) principle<sup>†</sup> have gained increasing popularity. The inversions of control concept means that not all the dependencies need to be known at compile time but may be resolved at runtime. The flow of a particular application is dictated by the framework itself, thus making the software highly configurable and extendable. Such extensibility can be achieved by overriding particular methods, or via a plug-in-based approach. In this case, the source code of the framework is not supposed to be modified by end-users.

Here, we aim to combine the domains of science and enterprise to create a platform for performing computational chemistry in modern computing environments. To do so, we have developed JACOB by applying rigorous software engineering techniques to make extensible and testable code. A framework is abstract by the fact that it provides only a partial solution; in the case of JACOB, one still needs to add the scientific code, only then does the framework lose its abstract nature and become part of a complete functioning application. This abstraction gap enables the framework’s design to be generically applicable, and because the framework was not constructed for any particular specific problem, no specific implementations are presented here. The reader is actively encouraged to envisage his or her own possible use case scenarios. Applications that have already been built on the JACOB framework are reported elsewhere for the interested reader.<sup>[23,24]</sup>

## Implementation

Object orientated programming (OOP) was developed in the 1960s to improve the modularity and scalability of a code base. In OOP, the class takes center stage, where real-world objects from a particular problem domain are mapped on to a set of classes that contain data fields (variables) and methods (procedures). Object orientation helps to create flexible and more manageable code that is built from many relatively uncoupled classes. The classes in the JACOB framework adhere to the enterprise bean convention<sup>[25]</sup> and are either plain old java<sup>[26]</sup> objects or their dynamic descendant, plain old groovy<sup>[27]</sup> objects (POGOs). Spring<sup>[28]</sup> is an application devel-

opment framework for enterprise Java. The Spring framework is responsible for the creation of the “Spring beans” inside the IoC container. The process of defining these relationships between the Spring beans (such as dependencies) is known as wiring. The wiring in JACOB is configured using a number of different technologies. The infrastructure layer is wired using the traditional Extensible Markup Language (XML)-based configuration. An annotation-based configuration is used for simple beans. Finally, complicated relationships between beans are programmatically configured inside a series of configuration classes. The dependent elements (such as objects or values) are injected into the destination classes automatically according to the wiring specified at run-time. The design pattern is also known as dependency injection and has the added advantage that it makes code more amenable to unit testing.

JACOB adopts the concept of coding to interfaces throughout to facilitate extensibility. The components used to create a single execution of the JACOB framework are therefore highly interchangeable. Implementations for future application code should be written against the interfaces provided by JACOB’s application programming interface (API), when possible. We note that modern integrated development environments (e.g., Spring Tool Suite (STS),<sup>[29]</sup> Eclipse,<sup>[30]</sup> NetBeans,<sup>[31]</sup> IntelliJ IDEA<sup>[32]</sup>) make this extremely easy as stubs for all required methods are automatically created for the developer when implementing a particular interface. The JACOB framework can also be extended via plugins when a suitable interface does not already exist. Abstract classes are also used wherever possible to reduce the size of the code base, while increasing the productivity of development efforts.<sup>‡</sup> Clean and transparent coding practices were applied throughout JACOB for clarity of future developers.<sup>[33]</sup> Documentation that describes the dependencies of classes and their methods is automatically generated out of the source code using Groovydoc.<sup>[34]</sup> In Java, it is trivial to create java interfaces to other traditional compiled languages such as Fortran, C, or C++, for computationally intensive tasks.

In accord with the domain-driven design pattern, the JACOB framework employs a strict separation of concerns, by employing a number of layers:

- **Model:** In the domain model pattern, objects that are persisted are represented as a set of domain models. Therefore, the model layer contains a set of objects that hold the current state of an application, that is, the data.
- **Service:** A service is simply a class (or set of classes) that contains code that performs application specific logic. In a scientific application, this would typically include algorithms. Services can be used to change the state of the models, that is, procedures.
- **Data access object (DAO):** Data access objects are specifically used for persistence. This means that the persistence layer can be decoupled from the service and data layers.
- **AOP:** Aspect-orientated programming handles the secondary functionality introduced by “cross-cutting” elements.

<sup>†</sup>The inversion of control is often described colloquially as the “Hollywood principle” which has arisen from the “Don’t call us, we’ll call you” phrase. This is a form of implicit invocation.

<sup>‡</sup>Adhering to the “Don’t Repeat Yourself” (DRY) principle of software design.

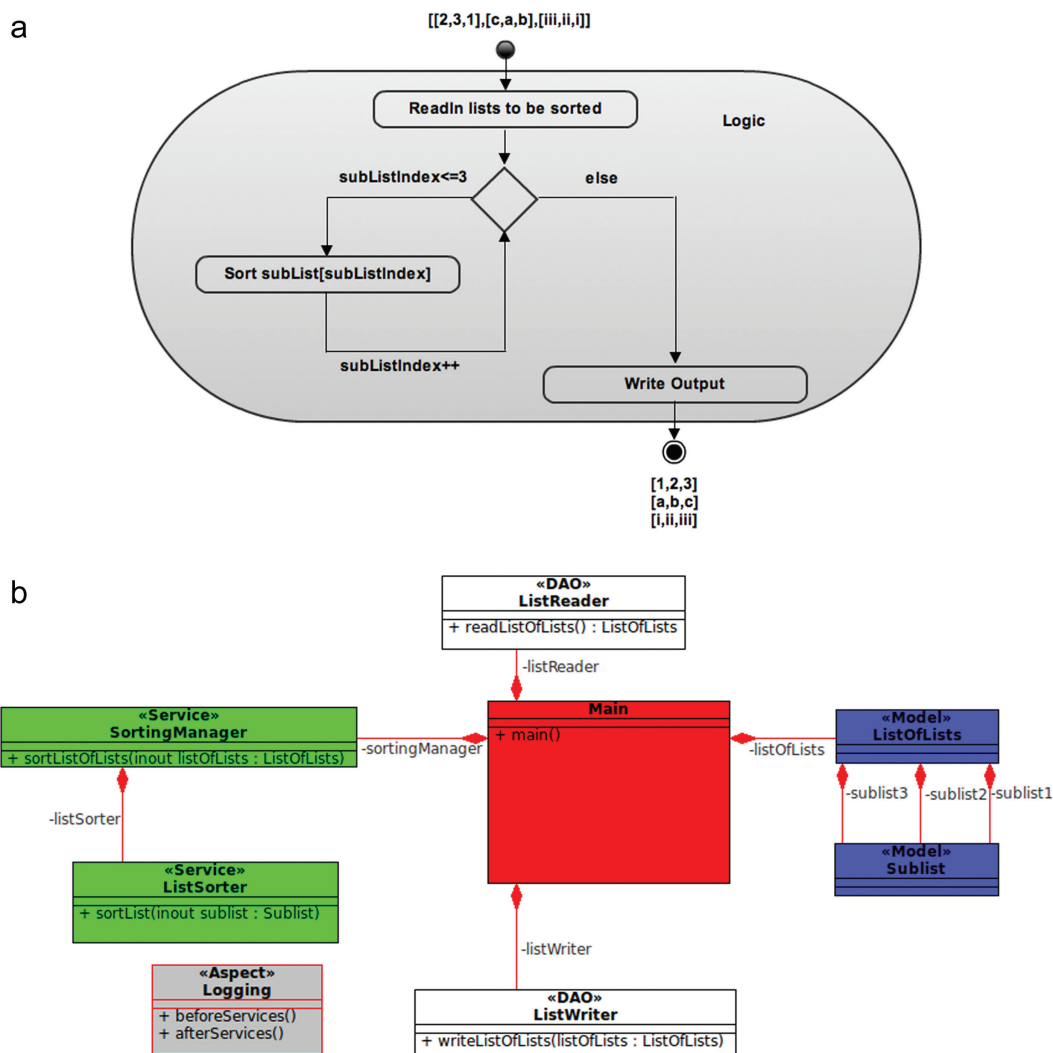


Figure 1. a) an activity diagram and b) domain model diagram [Models (yellow), services (blue), DAOs (green), Aspect (grey), and the Main class (red)] for an illustrative example on how one might map from a workflow to a domain model that is suitable for JACOB.

Cross cutting elements make it difficult to adhere to the pure object orientation paradigm. Aspects are methods that are called before or after the execution of other methods.<sup>[35,36]</sup> Because aspects are not located in the classes they work on, it is therefore possible to decouple their functionality from where it is needed. Aspects can therefore be centralized, added, removed, or changed without changing other parts of the framework.

An illustrative example of the mapping required from a workflow (procedural) to a domain model (object oriented) that is needed for the JACOB framework is shown in Figure 1.

The master–slave architecture was chosen for JACOB due to the nature of computational chemistry where the processing of jobs is typically the bottleneck, and spreading jobs across multiple worker-nodes is often desirable. The master layer is used primarily for user interaction, job setup, interacting with the results, and so on. The slave layer is for computational intensive tasks where the models and services are typically needed to solve scientific problems. There is typically a one-to-many relationship to the slave compute nodes. An overview of the framework infrastructure is given in Figure 2.

Importantly, the ability to select each component, configuring the location of each component, and configuring the communication between the components, is an advantage of using a modular and flexible framework. These options are left to the application developer based on their particular requirements. For instance, if scalability is not an issue, then the slave and master layers of JACOB do not need to be on separate computers. The location of the database holding the results also does not need to be on the master node. The location of the middleware is also configurable.

The configuration of the master and slave layers can be changed using profiles. A profile is a set of user definable configuration options. These profiles can be set depending on the current environment; this efficiently enables multiple “default” configurations of the framework. An environment is an intentionally abstract term that may include production, test, or development. A production environment needs stability, speed, and persistence. On the other hand, in a development environment, it is not convenient to connect to a production database, for example, this may corrupt valuable data, or cause

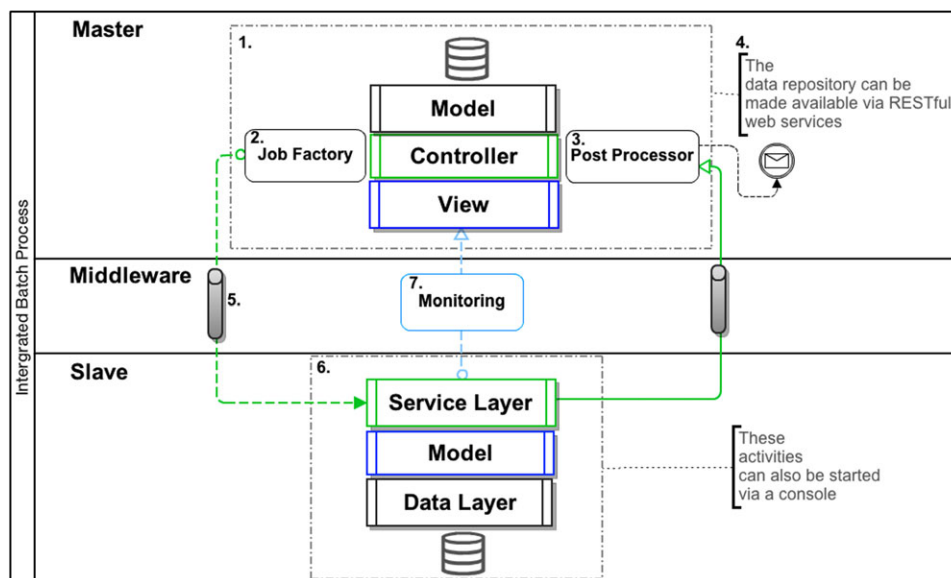


Figure 2. The architecture of the JACOB framework is multilayered and is connected using middleware. [Color figure can be viewed in the online issue, which is available at [wileyonlinelibrary.com](http://wileyonlinelibrary.com).]

significant delays in the developer time while waiting for the production database to be bootstrapped. The test environment is typically a clone of the production environment, which is useful to reduce the chances of bugs being introduced into production. Only after code is thoroughly tested, for example, performing scientific computation using integration tests, should the production environment be used.

In distributed computing systems, there exists some core systemic requirements namely consistency, availability, and partition tolerance. Consistency meaning that the nodes see the same data, availability means that the system is fully operational, and partition tolerance means that a distributed system can function even when internode communication is severed. Brewer's theorem<sup>[37]</sup> states that only two of these requirements can be fully satisfied, and in distributed computing a compromise must be made. Because JACOB is a modular framework, the application developers can make decisions on where to compromise based on their own specific requirements for a given application or environment.

### Master

The web tier of JACOB is developed on the Grails<sup>[38]</sup> dynamic web framework that uses the model-view-controller design pattern.

The three main components are:

a. **Model:** the model layer contains a set of objects that hold the current state of an application. A given problem in computational chemistry can be mapped onto an appropriate set of domain models. For example, a job domain model may contain input parameters (e.g., density functional and basis set) and data parsed from input files (e.g., molecular structure). A relational database schema is set up based on the set of domain models. After the job has been processed, the results may be stored in a domain model (e.g., energy and optimized structure) and may be persisted into a database.

b. **Controller:** the controller conveys the message between the model and view. In particular, it converts a map of parameters from user inputs into the model. The controller is also responsible for retrieving data from the model and returns it to the user, for example, as html, XML, or JSON. More computationally demanding tasks can be implemented as services. Services can then be injected into the controllers for post processing, for purposes such as data-mining.

c. **View:** users interact with the framework through the view layer. The view renders a webpage. The webpage can contain a mixture of static html and dynamic content that is

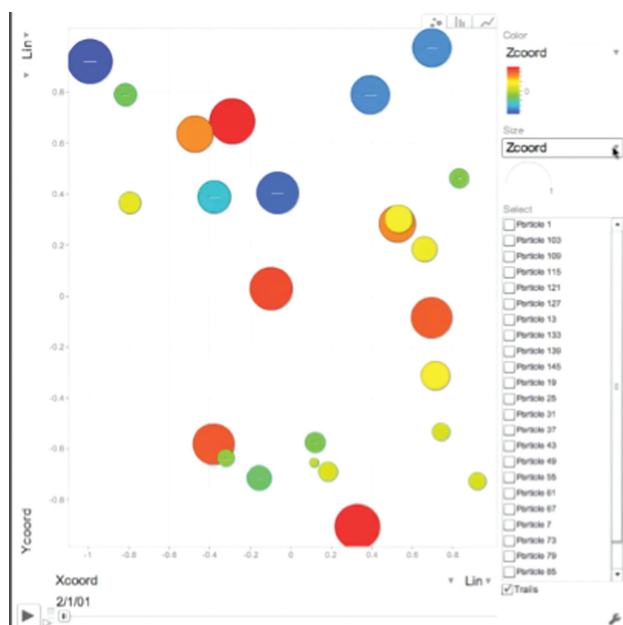


Figure 3. The JACOB framework can render a dynamic graph (Google visualization API) for post processing of job results.



retrieved from the model by the controller. Information relating to the current state of the framework can be queried by using search capabilities for selected domain models.

Separating the concerns of the web-application layer reduces the amount of code and its complexity. Typically, an application developer creates the domain models and the services, as these are both domain-specific problems and therefore cannot be handled by a generic framework. The Grails framework then automatically scaffolds a lot of the “boilerplate” code needed for the view and controller layers.

### Job factory web flow

The dynamic web framework automatically handles various repetitive tasks that are prone to errors from manual handling. For example, transferring jobs across multiple nodes, job queuing, job execution, as well as post job data collation. A calculation performed with JACOB will be carried out as following:

- STEP 1: A user may select a specific algorithm for his/her problem from the view page.
- STEP 2: The user can then give the appropriate job parameters for the corresponding algorithm. These parameters will then be persisted into the model layer.
- STEP 3: A HTML5 drag and drop file handler is provided for uploading the input files used for the calculation. A job can only be validated and started if an input file has been uploaded to the application, if appropriate.
- STEP 4: Once a complete job is defined, a message is sent to a message-based middleware broker. A slave job listener (see below) is installed on the compute cluster (node), and is used to receive jobs from the queue. A cluster node will accept, and start a new job in queue (if any), once it becomes free.

### Job post processor

When the calculation is finished, the results will be sent back to the web application and persisted into the database for subsequent analysis or dissemination. A data visualization controller has been implemented for post processing the results of the calculation. The data visualization controller is responsible for retrieving results from the database, performing statistical analysis services, and redirects the analyzed results to the corresponding view page.

The user can interact with the calculation results in different ways:

1. Browse lists of completed jobs.
2. Perform a query against the completed jobs.
3. Download a log file for the job for further analysis, with the HTML5 file handler.
4. Visualize molecular structure with a Jmol applet.<sup>[39]</sup>
5. Job results can be visualized using the API from Google’s visualization library. An example of such graph is shown in Figure 3.

### Web services

A RESTful web API is used in the JACOB framework to enable data exchange between remote systems. Web services are designed for intermachine communication. REST stands for representational state transfer, and it is an architectural style whereby a client requests a resource from a server using a predefined method. The methods include: GET (retrieves a resource), POST (creates a resource), PUT (updates a resource), and DELETE (deletes the resource). The resource is described by its unified resource identifier. The master layer of JACOB acts as the server and exposes the web services, remote clients then submit a request, and the response is returned from the JACOB server, see Figure 4.

Spring-security<sup>[40]</sup>-based permissions are used in JACOB to provide a simple mechanism for configuring accessibility to the data repositories. Allowing any user to make a DELETE method call on your scientific results is obviously undesirable. The RESTful web services are language interoperable, and the return format of the message can requested to be in XML, JSON, plain text, and so on.

A user interface is provided via a dynamic web application (Web 2.0), this enables a user to be directed through the application to a specific point via the URL mappings. The end of the URL contains two pieces of information: (a) the domain model from which requested data can be retrieved from or stored into, and (b) a controller action, which specifies the user’s intentions with the requested data.

For example:

`http://.../job/create/`

This directs the user to the page where a user can create (controller action) a new job. The corresponding job parameters that the user has entered will be stored in the job domain model, which can be subsequently viewed:

`http://.../job/show/1`

This URL returns a show (controller action) page for job with ID 1 that contains information stored in the run domain model for the corresponding ID.

`http://.../datavisualization/motionchart?job=2`

This directs to a view page that calls the action of plotting a motion chart from the data visualization controller. The data to be plotted should be retrieved from job ID 2.

Therefore, by understanding the URL mappings, and having some knowledge of the data structure, a remote user can interact with the contents of the database via the URLs. However, the view layer can include appropriate hyperlinking, which makes this interaction simpler for the users who do not have interment knowledge of URL mappings or the data structure.

### Message-orientated middleware

The implementation of message-oriented middleware (MOM) such as the apache active MQ<sup>[41]</sup> enables loose coupling between the master and slave layers. The master and slave layers therefore do not directly communicate, instead messages are given to a message broker. The messages are stored on the broker until a request is made to process the message.

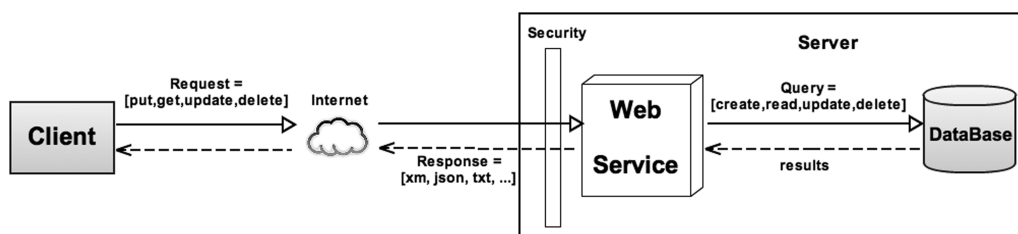


Figure 4. A schematic of the REST web service architecture.

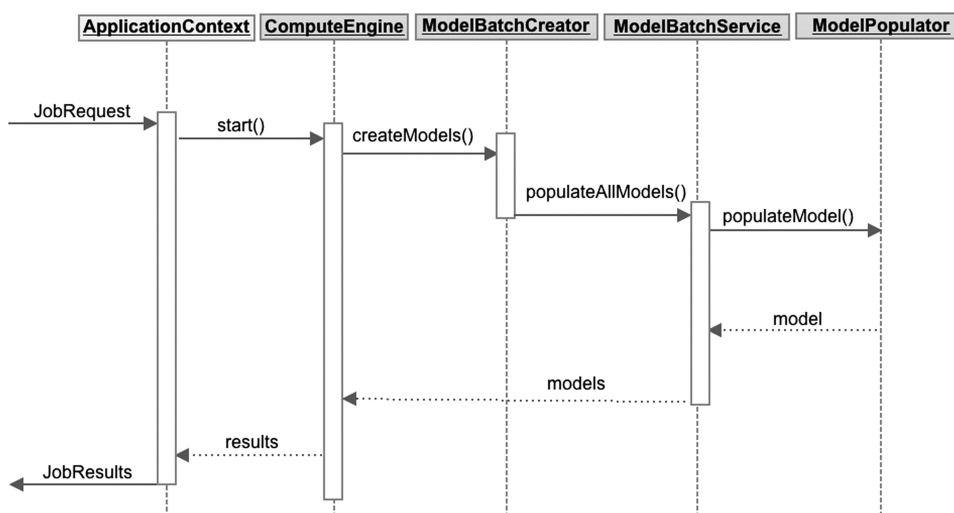


Figure 5. Sequence diagram for the slave tier of the JACOB framework. Interfaces are shown as gray boxes, method calls are on filled arrows, and objects are returned on dotted lines.

This enables the layers of the systems to communicate asynchronously. Using this paradigm, a form of load balancing is trivially introduced. If the master fails, the responses from the slave nodes will be queued, and only consumed by the master when it is made available again. On the other hand, if the slave(s) fails, the messages will be queued until a slave is again ready to consume the queued requests. The final advantage of this approach is that delivery is guaranteed by using transactions, which increases the robustness of our solution stack.<sup>5</sup> The MOM handles the master–slave nature of the client/server mechanism, leaving the more interesting service layer for the future application developers. Of note, a direct connection can be made between the slave nodes and the master if required, for example, via SQL connections.

### Slave

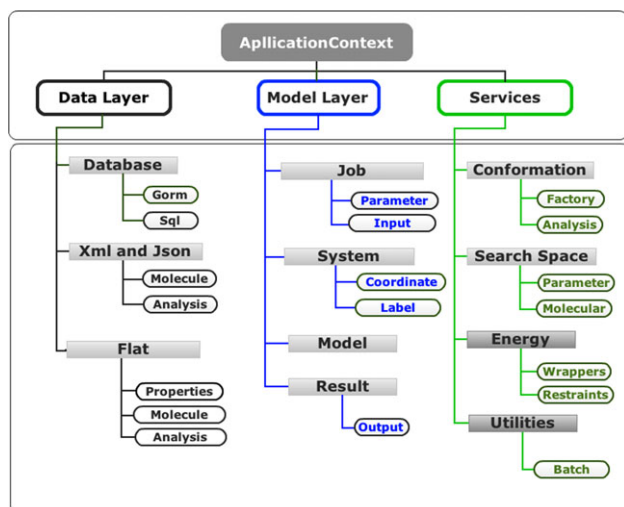
The operational flow of the slave layer is given in Figure 5. A *jobRequest* object is received from the MOM, and then the *applicationContext* is built. Only the beans needed for a given job are created in the Spring container. The framework uses a batch-based approach to process a set of models for a specific configuration of the application. We have implemented a se-

quential (nonthread safe) simple batch service, while for a more robust and efficient implementation, a scalable batch service from Spring batch<sup>[42]</sup> partitions the work into “chunks.” The logical flow for a specific computational chemistry job is driven by a *computeEngine*. More specifically, before a job is run, the engine performs preprocessing tasks and calls the batch creator class to generate a set of models. The batch of domain models are then processed on the slave. For example, a set of molecules may be represented by a set of molecule domain models each containing an energy field. The processing of this set includes computing single-point energies and populating the domain models with the parsed energies from the external quantum chemical code wrappers. The *computeEngine* is also capable of post processing the models at the slave level, and then the results are sent back to the centralized master and collated as *jobResults*.

The slave component is also separated into layers: this enables a convenient and practical separation of concerns. Some key components are given in Figure 6. More specifically:

a. Model layer is used to store the state of objects that are needed for processing a particular job. These set of objects, and their relationships, are defined in POGOs, which are mapped onto a corresponding relational database schema, in the same fashion as on the master layer. For example, a *molecularSystem* is a domain model that stores all of the information that defines the system under investigation, including initial coordinates, atomic labels, and number of molecules.

<sup>5</sup>A ‘solution stack’ refers to the complete set of software components that are used to fulfill the requirements of the system.



**Figure 6.** An overview of the important components of the slave tier within the JACOB framework. The application context defines the runtime of the JACOB framework. [Color figure can be viewed in the online issue, which is available at [wileyonlinelibrary.com](http://wileyonlinelibrary.com).]

b. Data layer is for persistence within the framework and a diverse range of DAO are available (Fig. 6). An object relational mapping between the domain models and the database is carried out using Gorm<sup>[43]</sup> (Hibernate<sup>[44]</sup>). A traditional structure query language (Sql) DAO is also available for simple create, read, update, and delete operations. Chemical markup language-based XML files can be parsed by the *XmlCoordinateReader* to populate a molecular system. A standard key value pair properties file can be used locally to configure the application, which enables the framework to configure an application in a flexible manner and requires no changes to the code.

c. Services constitute the largest component of the framework. A range of generic tools such as conformationFactories for creating conformations and a set of generic structure analysis tools are included in JACOB. Wrappers for a number of quantum chemical programs (e.g., Mopac,<sup>[45]</sup> Orca,<sup>[46]</sup> Gaussian,<sup>[47]</sup> Turbomole<sup>[48]</sup>) are available and further wrappers are easy to implement. A custom citation annotation is provided for tagging classes that implement algorithms from the scientific literature. A second custom annotation is implemented for tagging classes that wrap external software that details their licensing requirements. Overall, this allows all relevant citations and licenses to be printed out automatically, if, and only if, a particular class had been used in the current configuration of the application.

### Monitoring

The AOP of JACOB handles tasks such as logging<sup>[49]</sup> and security,<sup>[50]</sup> and monitoring which are examples of secondary functionality that are needed in many different places of an application. Java management extensions (JMX) enables one to modify code (such as adjusting parameters) that is currently running. This is an additional feature that is not normally available for traditional computational chemistry languages such as Fortran or C, where recompilation is required. To make remote

modifications to the code, JMX has the ability to monitor the code on-the-fly. This allows data collected from the running application to be used to make informed operational decisions. Due to the nature of batch programming, large datasets can be distributed among many slaves and therefore a monitoring tool is highly desirable.

### Validation

A suite of unit and integration tests have been written to cover a large percentage of the code base using the Spock testing and specification framework.<sup>[51,52]</sup> Unit tests cover only one particular class, while integration tests are aimed at testing whether multiple classes function together correctly. The test code coverage of the JACOB framework was monitored using the Clover tool.<sup>[53]</sup> The Gradle<sup>[54]</sup> build automation tool was used to compile, package, manage dependencies, and create the documentation. The set of third party dependencies were automatically synchronized with the Maven central repository.<sup>[55]</sup> The Jenkins<sup>[56]</sup> continuous integration server to ensure code integrity during development. Continuous integration means the source code was automatically built and the suite of tests were run and logged whenever any code was checked in to the source code repository.<sup>[57,58]</sup> Performance was monitored using Spring Insight<sup>[59]</sup> stress testing framework in a VMware vFabric Tomcat server.<sup>[60]</sup> The CodeNarc<sup>[61]</sup> framework was used to monitor differences in coding practices among the members of the development team.

### Conclusions

Here, we have developed an integrated framework based on enterprise architecture to create a platform for performing computational chemistry within modern computing environments.

The following design goals were successfully implemented:

- **Extensibility:** Modern programming concepts were rigorously applied to create an easily extensible framework, leaving future developers more time to solve science-specific problems in a versatile manner.
- **Scalability:** A master–slave relationship allows dynamic scaling by allocating as many slaves as required for a given job. A centralized master makes monitoring trivial.
- **Testability:** Continuous integration with a unit and integration test suite ensures code integrity in a team-based development environment.
- **Accessibility:** User- and role-based authentication and access-control is implemented across the master layer.
- **Robustness:** The framework is built on top of well-established open source enterprise grade software components.

JACOB is a centralized user-friendly framework for job creation, processing, and analysis that is capable of handling large and complex datasets. Data repositories can be accessed using Web 2.0 technologies or RESTful web services. Therefore, the integrated enterprise framework can be used as an effective collaborative tool for alleviating organizational complexity (e.g., in widespread collaborative research, virtual laboratories, and chemical or pharmaceutical industries). Possible

applications of JACOB include systematic studies and/or complicated workflows, for example, in benchmarking, parameter searching, conformational searching, multiscale modelling, docking, or high throughput virtual screening.

**Keywords:** computational chemistry · batch · enterprise · framework · workflow

How to cite this article: M. P. Waller, T. Dresselhaus, J. Yang, *J. Comput. Chem.* **2013**, *34*, 1420–1428. DOI: 10.1002/jcc.23272

- [1] Top 500, Available at: <http://www.top500.org/lists/2012/06>, Accessed on: November 30, 2012.
- [2] G. E. Moore, *Electronics Mag.* **1965**, *4*.
- [3] Grisu, Available at <http://projects.arcs.org.au/trac/grisu/>, Accessed on: November 30, 2012.
- [4] H. Lederer, G. J. Pringle, D. Girou, M. -A. Hermanns, G. Erbacci, Parallel Computing: Architectures, Algorithms and Applications, In C. Bischof, M. Bücker, P. Gibbon, G. R. Joubert, T. Lippert, B. Mohr, F. Peters, Eds.; Jülich: NIC Series, **2007**, *38*, pp. 687–688.
- [5] D. A. Reed, *Comput. IEEE Comput. Soc.* **2003**, *36*, 62–68.
- [6] R. Pordes, D. Petravick, B. Kramer, D. Olson, M. Livny, A. Roy, P. Avery, K. Blackburn, T. Wenaus, F. Würthwein, I. Foster, R. Gardner, M. Wilde, A. Blatecky, J. McGee, R. Quick, *J. Phys.* **2007**, *78*, 012057.
- [7] M. Korth, S. Grimme, QMC@HOME, Available at: <http://qah.uni-muenster.de>. Accessed on November 30, 2012..
- [8] M. Korth, A. Lüchow, S. Grimme, *J. Phys. Chem.* **2008**, *112*, 2104.
- [9] BOINC, Available at <http://boinc.berkeley.edu>, Accessed on November 30, 2012.
- [10] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. P. Berman, P. Maechling, In 5th IEEE International Conference on e-Science, Oxford, United Kingdom, **2009**. Available at: <http://arxiv.org/abs/1005.2718>. Accessed on November 30, 2012.
- [11] T. McPhillips, S. Bowers, D. Zinn, B. Ludäscher, *Future Gener. Comput. Syst.* **2009**, *25*, 541.
- [12] M. R. Berthold, N. Cebron, F. Dill, T. R. Gabriel, T. Kçtter, T. Meinl, P. Ohl, C. Sieb, K. Thiel, B. Wiswedel, Studies in Classification, Data Analysis, and Knowledge Organization (GfKL 2007); Springer: Heidelberg, **2007**.
- [13] A. Tiwari, A. K. T. Sekhar, *Comput. Biol. Chem.* **2007**, *31* 305.
- [14] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, P. Li. *Bioinformatics* **2004**, *20*, 3045.
- [15] P. Sherwood, A. H. de Vries, M. F. Guest, G. Schreckenbach, C. R. A. Catlow, S. A. French, A. A. Sokol, S. T. Bromley, W. Thiel, A. J. Turner, S. Billeter, F. Terstegen, S. Thiel, J. Kendrick, S. C. Rogers, J. Casci, M. Watson, F. King, E. Karlsen, M. Sjøvøll, A. Fahmi, A. Schäfer, Ch. Lennartz, *J. Mol. Struct. (Theochem.)* **2003**, *632*, 1.
- [16] TCL, Available at <http://www.tcl.tk/about/language.html>, Accessed on November 30, 2012.
- [17] C. R. Jacob, S. M. Beyhan, R. E. Buló, A. S. P. Gomes, A. W. Götz, K. Kiewisch, J. Sikkema, L. Visscher, *J. Comput. Chem.* **2011**, *32*, 2328.
- [18] Python, Available at <http://www.python.org/>, Accessed on November 30, 2012.
- [19] WebMo, Available at <http://www.webmo.net/>, Accessed on November 30, 2012.
- [20] A. R. Allouche, *J. Comput. Chem.* **2011**, *32*, 174.
- [21] G. Hohpe, B. Woolf, Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions; Addison-Wesley: Boston, **2004**.
- [22] J. Yang, M. P. Waller, *J. Chem. Inf. Model.* **2012**, *52*, 3255.
- [23] J. Yang, M. P. Waller, *J. Phys. Chem. A.* **2013**, *117*, 174.
- [24] T. Dresselhaus, J. Yang, S. Kumbhar, M. P. Waller, *J. Chem. Theory. Comput.* (in press). <http://dx.doi.org/10.1021/ct301079m>.
- [25] Available at <http://www.oracle.com/technetwork/java/javaee/ejb/index.html>, Accessed on November 30, 2012.
- [26] JDK7, Available at <http://jdk7.java.net/>, Accessed on November 30, 2012.
- [27] Groovy, Available at <http://groovy.codehaus.org/>, Accessed on November 30, 2012.
- [28] Spring, Available at <http://www.springsource.org/spring-framework>, Accessed on November 30, 2012.
- [29] Spring Tool Suite, Available at <http://www.springsource.org/sts>, Accessed on November 30, 2012.
- [30] Eclipse, Available at <http://www.eclipse.org/>, Accessed on November 30, 2012.
- [31] NetBeans, Available at <http://netbeans.org/>, Accessed on November 30, 2012.
- [32] IntelliJ, Available at <http://www.jetbrains.com/idea/>, Accessed on November 30, 2012.
- [33] R. C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship; Prentice Hall International, Westford, Massachusetts, United States of America, **2008**.
- [34] GroovyDoc, Available at <http://gradle.org/docs/current/dsl/org.gradle.api.tasks.javadoc.Groovydoc.html>, Accessed on November 30, 2012.
- [35] CGlib, Available at <http://cglib.sourceforge.net/>, Accessed on November 30, 2012.
- [36] AOP Alliance, Available at <http://aopalliance.sourceforge.net/>, Accessed on November 30, 2012.
- [37] N. Lynch, S. Gilbert, *ACM SIGACT News* **2002**, *33*, 51.
- [38] Grails, Available at <http://grails.org/>, Accessed on November 30, 2012.
- [39] Jmol: an open-source Java viewer for chemical structures in 3D, Available at <http://www.jmol.org/>, Accessed on November 30, 2012.
- [40] Spring Security, Available at <http://www.springsource.org/spring-security>, Accessed on November 30, 2012.
- [41] ActiveMq, Available at <http://activemq.apache.org/>, Accessed on November 30, 2012.
- [42] Spring Batch, Available at <http://www.springsource.org/spring-batch/>, Accessed on November 30, 2012.
- [43] Grails, Available at <http://grails.org/doc/latest/guide/GORM.html>, Accessed on November 30, 2012.
- [44] Hibernate, Available at <http://www.hibernate.org/>, Accessed on November 30, 2012.
- [45] J. J. P. Stewart, MOPAC2012, Stewart Computational Chemistry; Colorado Springs, CO, USA, Available at <http://OpenMOPAC.net>, Accessed on November 30, 2012.
- [46] F. Neese, *WIREs Comput. Mol. Sci.* **2012**, *2*, 73.
- [47] M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, G. Scalmani, V. Barone, B. Mennucci, G. A. Petersson, H. Nakatsuji, M. Caricato, X. Li, H. P. Hratchian, A. F. Izmaylov, J. Bloino, G. Zheng, J. L. Sonnenberg, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, J. A. Montgomery, Jr., J. E. Peralta, F. Ogliaro, M. Bearpark, J. J. Heyd, E. Brothers, K. N. Kudin, V. N. Staroverov, R. Kobayashi, J. Normand, K. Raghavachari, A. Rendell, J. C. Burant, S. S. Iyengar, J. Tomasi, M. Cossi, N. Rega, J. M. Millam, M. Klene, J. E. Knox, J. B. Cross, V. Bakken, C. Adamo, J. Jaramillo, R. Gomperts, R. E. Stratmann, O. Yazyev, A. J. Austin, R. Cammi, C. Pomelli, J. W. Ochterski, R. L. Martin, K. Morokuma, V. G. Zakrzewski, G. A. Voth, P. Salvador, J. J. Dannenberg, S. Dapprich, A. D. Daniels, Ö. Farkas, J. B. Foresman, J. V. Ortiz, J. Cioslowski, D. J. Fox, Gaussian 09, Revision A.1; Gaussian, Inc.: Wallingford, CT, **2009**.
- [48] TURBOMOLE V6.4 2012, a development of University of Karlsruhe and Forschungszentrum Karlsruhe GmbH, 1989–2007, TURBOMOLE GmbH, since **2007**; available at [www.turbomole.com](http://www.turbomole.com) Accessed on November 30, 2012.
- [49] Log4j, Available at <http://logging.apache.org/log4j/2.x/>, Accessed on November 30, 2012.
- [50] Spring Security, Available at <http://www.springsource.org/spring-security>, Accessed on November 30, 2012.
- [51] Spock, Available at <http://spockframework.org>, Accessed on November 30, 2012.
- [52] T. Strassner, M. Busold, W. A. Herrmann, *J. Comput. Chem.* **2002**, *23*, 282.
- [53] Clover, Available at <http://www.atlassian.com/software/clover/overview>, Accessed on November 30, 2012.
- [54] Gradle, Available at <http://www.gradle.org/>, Accessed on November 30, 2012.



- [55] Mvn repository, Available at <http://mvnrepository.com/>, Accessed on November 30, 2012.
- [56] Jenkins, Available at <http://jenkins-ci.org/>, Accessed on November 30, 2012.
- [57] Bit Bucket, Available at <https://bitbucket.org/>, Accessed on November 30, 2012.
- [58] Git Hub, Available at <https://github.com/>, Accessed on November 30, 2012.
- [59] Spring Insight, Available at <http://www.springsource.org/insight/>, Accessed on November 30, 2012.
- [60] VMware vFabric tc Server Developer Edition v2.7, Available at <http://www.vmware.com/>, Accessed on November 30, 2012.
- [61] CodeNarc, Available at <http://codenarc.sourceforge.net/>, Accessed on November 30, 2012.

---

Received: 2 January 2013  
Revised: 15 February 2013  
Accepted: 18 February 2013  
Published online on 3 April 2013

---