A Database-Centric Virtual Chemistry System

Peter Lind* and Markus Alm

Medivir AB, Lunastigen 7, 144 44 Huddinge, Sweden

Received September 1, 2005

We describe an Oracle database application for general use within virtual chemistry. The application functions as a central hub and repository for chemical data with interfaces to external calculators. It deals with the general problems of merging data from disparate sources and with scheduling of computational tasks for parallel or sequential execution in a mixed environment. The central database is used for the storage of input, intermediary, and final data as well as for job control. A calculation job is split into distinct tasks, or units of work, which are put in a queue. Tasks are dequeued and handled by specialized calculators. These calculators are in-house or commercial programs for which adaptor modules for connection to the database must be written. Tasks are handled in a transactional fashion, so that uncompleted or failed tasks are left in the queue. This makes the system stable to many types of disturbances. Sorting, filtering, and merging operations are handled by the database itself. Usage is very general, but some specific examples are (1) as a back end for a chemical property calculator Web page, (2) in an automated quantitative structure−activity relationship system, and (3) in virtual screens.

INTRODUCTION

A central mission of chemoinformatics is to find efficient means to automate workflows involving the processing of chemical data. Data sets may need to be assembled from different sources, come in different formats, and exist on disparate platforms. Often, workflows are such that data sets are transferred and processed in a chain of many steps from one program to another. As an example, it is common in quantitative structure−activity relationship (QSAR) studies that compounds and activity data are first assembled from a database and then stored in a file to be used by a property calculator program, the output of which is fed into a statistical or regression package for final analysis. The process becomes more complicated if one wishes to include and merge properties from different sources.

One common way to achieve some degree of automation in these workflows is to use scripts or batch files.

Another approach is to use special pipelining software.[1,2] The pipelining paradigm models the overall computational process as data flowing through a network of modular computational components. The software implementation of a pipelining system may use commercial or in-house programs for some of these components. Data pipelining is described as a technology which is complementary to relational database systems.[3]

We have tried an automation approach which is centered around a database, and we discuss our solution in this paper. This approach models a workflow as computational components acting on a data set. We view chemical data as the central entity in a chemoinformatics workflow and think of the commercial and in-house programs that we want to integrate in our workflow as creators of such data. They are actors that take various actions on a data set, often augmenting the data set with new kinds of data. The implementation uses packages and tables in an Oracle[4] relational database management system (RDBMS). Oracle databases can be implemented on all popular operating systems, and the computational components can be distributed over different platforms.

The following critical properties of RDBMSs motivate their use in this context:

(1) Many external processes can connect to the database and act concurrently on the same data set with predictable results. This is useful when a computational task is trivially parallelizable; that is, the task can be divided into subtasks, which can be executed independently from each other. This is often the case in chemoinformatics; for example, the log *P* of some individual compounds in a set can be calculated separately and independently from the others. Many tasks of different types can also be performed simultaneously, such as running docking scores and log *P*.

(2) RDBMSs are optimized for the sorting, filtering, and merging of data sets of practically any size. These are very common operations in chemoinformatics.

(3) Data are inserted and updated in transactions; that is, a set of changes to data either fails or succeeds as a unit. In other words, we can make sure that no data are left only partly processed. The transactional property is critical in a distributed computing environment because it provides for protection from data corruption in the event of network or process faults. A failed transaction will be aborted and its effects on data undone.

(4) There are several well-known database connectivity application programming interfaces (APIs) that work well on all common platforms and over most networks. There is no need to restrict calculations to any single operating system.

The use of RDBMSs is by no means new to chemoinformatics—they have been widely used for corporate structure and test result databases and for chemical inventories,[5−7]

* Corresponding author phone: 46 8 608-3126; fax: 46 8 608-3199; e-mail: peter.lind@medivir.se.

10.1021/ci050360b CCC: $33.50 © 2006 American Chemical Society
Published on Web 03/16/2006

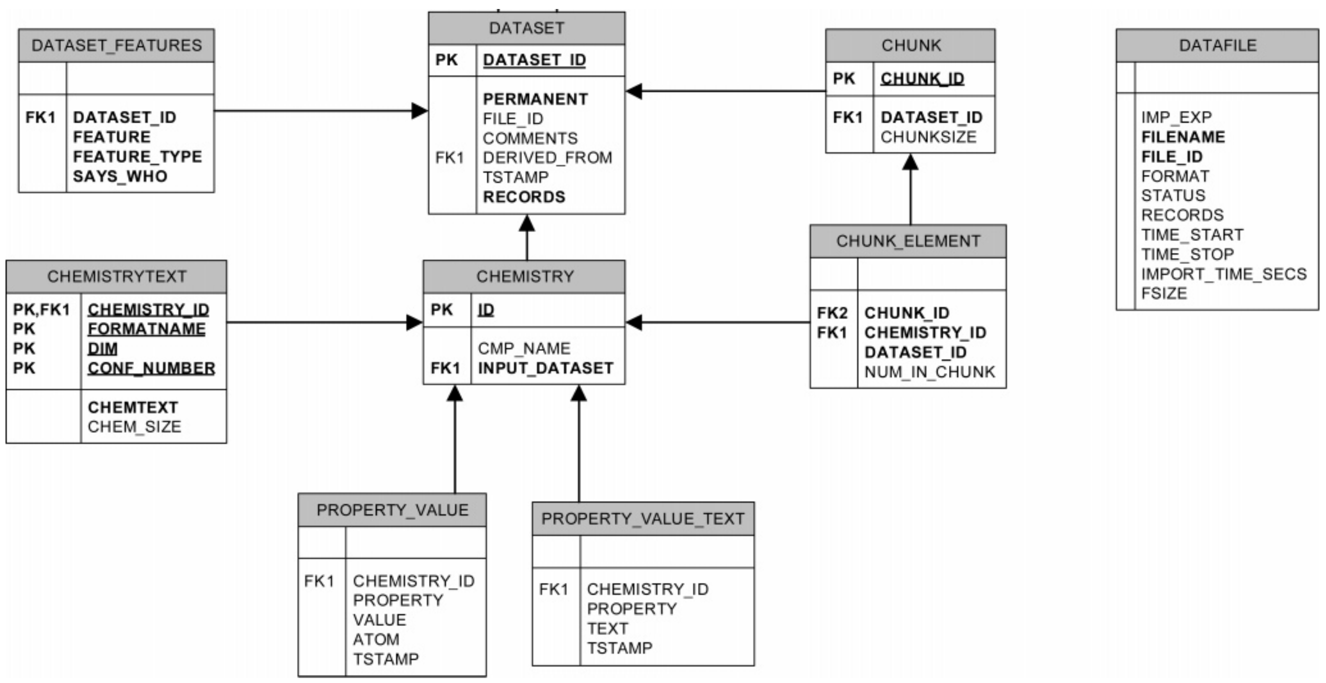
DATASET_FEATURES
FK1 DATASET_ID FEATURE FEATURE_TYPE SAYS_WHO
DATASET
PK DATASET_ID
PERMANENT FILE_ID COMMENTS FK1 DERIVED_FROM TSTAMP RECORDS
CHUNK
PK CHUNK_ID
FK1 DATASET_ID CHUNKSIZE
DATAFILE
IMP_EXP FILENAME FILE_ID FORMAT STATUS RECORDS TIME_START TIME_STOP IMPORT_TIME_SECS FSIZE
CHEMISTRYTEXT
PK,FK1 CHEMISTRY_ID PK FORMATNAME PK DIM PK CONF_NUMBER
CHEMTEXT CHEM_SIZE
CHEMISTRY
PK ID
CMP_NAME FK1 INPUT_DATASET
CHUNK_ELEMENT
FK2 CHUNK_ID FK1 CHEMISTRY_ID DATASET_ID NUM_IN_CHUNK
PROPERTY_VALUE
FK1 CHEMISTRY_ID PROPERTY VALUE ATOM TSTAMP
PROPERTY_VALUE_TEXT
FK1 CHEMISTRY_ID PROPERTY TEXT TSTAMP

Figure 1. Tables of the data repository. PK = primary key, FK = foreign key.

sometimes combined with screening[8] or decision-support tools[9] or other special features.[10,11]

In the next section, we will describe the key parts of an application that splits computational jobs into tasks which are distributed to calculators. By "application" we will mean the database objects that are available to be reused as new calculators and job types are added. We will not describe a full working system, although an example calculator is described in an outline. By "workflow" we mean a series of computational tasks to be performed. We use the word "job" for two things: (1) instructions stored to execute a workflow on a particular data set and (2) the processes executing such a job. By "job type" we mean instructions to execute a certain type of workflow. Job types are not part of the application; they are best written as Oracle PL/SQL procedures.

METHODS

Platforms and Programs. The application was implemented as an Oracle 9.2 database running on a Windows 2000 Server machine with a 3 GHz Intel Xeon processor and 1 GB of RAM.

Overall Structure of the Application. There are four logical parts: the chemical data repository, the calculator dictionary, the job control system, and the task queue system. All computational work is done by external calculators, which are not part of the application. The calculators interface the task queue and have insert privileges for the data repository. If the calculator is an external program, then an adaptor module must be written. The adaptor can be written in any language that supports one of the standards for database connectivity, for example, Java, C++, or Visual Basic. It should be possible to connect most programs that have some kind of automation interface. A key feature is that many calculators of the same or different types can be connected and run simultaneously. Each calculator will connect in a session of its own. The application was originally designed to serve as a back end to a property calculator intranet page,[12,13] with calculators for format conversion and for property calculation, but it should be possible to reuse most of the code and tables for any type of calculator that takes chemical data sets as input. Examples of potential calculators are programs for clustering, docking, 3D converters, fingerprinting, combinatorial enumeration, QSAR analysis, and pharmacophore searching and identification. Jobs can be launched and queried for status through an interface to the job control system. This interface is intended to be used from a Web page or directly from SQLPLUS or PL/SQL scripts.

We will next give an overview of the tables and code. Details regarding indexing and constraints will not be included.

Chemical Data Repository. This involves nine tables, as illustrated in the entity-relationship diagram shown in Figure 1. The table DATAFILE keeps track of host operating system files with I/O data. These may be user files containing data to be imported into the database or output files with structures or properties or both. They may also be intermediary files which are of no interest to the end user but which contain input or output data for any of the calculator programs.

All chemical structures belong to a unique input data set. A data set is typically imported from a file but can be created by a calculator or imported over a database link from another database. The table DATASET is a list of all data sets. A data set contains zero or more compounds with identifiers stored in the CHEMISTRY table. The primary key of that table (ID) is a number taken from an Oracle sequence. The CMP_NAME column is intended for the storage of identifiers from input files; these names are not constrained unique. Each compound has zero or more structure representations, which are stored in the CHEMTEXT column of table CHEMISTRYTEXT. There are two reasons for allowing multiple entries for the same compound in this table: (1) We may want to store the compound in several formats because different external programs or calculators work with

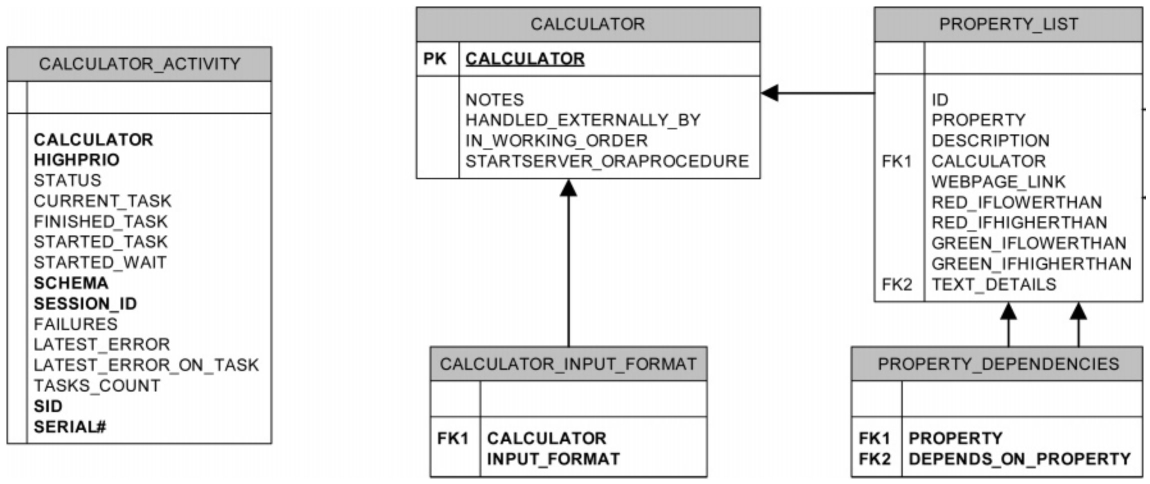

Figure 2. Tables of the calculator dictionary.

different formats. In practice, at least one of the calculators must be a format converter. (2) We may want to store several conformations of the same compound. The combination of format, dimensionality (2D or 3D), and conformation number is constrained unique.

An alternative approach would make use of a separate CHEMISTRY_CONFORMATION table referencing the CHEMISTRY table and would have result and structure representation tables reference individual conformations in that table. This arrangement should be better if conformation-dependent properties are to be studied, but it was not chosen here.

A data set is divided into one or more chunks. The table CHUNK stores identifiers for each chunk of a data set, and the CHUNK_ELEMENT table lists the individual compounds in each chunk. Chunking is an essential part of the design, and each calculator instance will only work on one chunk at a time. Any process importing a data set must call a CHUNKDATASET PL/SQL procedure. Even data sets containing a single structure are formally chunked for consistency; the overhead with this is negligible.

Any numeric properties of compounds are stored in the PROPERTY_VALUE table. These may be, for example, test results, estimated physical properties, or docking scores. The table PROPERTY_VALUE_TEXT contains any text messages from calculators. This table can be used by a Web page to display messages from calculators giving textual output. The table DATASET_FEATURES is a dictionary of the chemical formats and property types present in the data set at any moment. Features are inserted into this table as calculations proceed.

Calculator Dictionary. Calculators known to the application are listed in the CALCULATOR table shown in Figure 2. A calculator may be capable of producing many properties, and the table PROPERTY_LIST keeps information about what properties each calculator can output. Property names are unique in this table to enforce different naming when several calculators produce the same kind of property. There may, for example, be X_LOGP and Y_LOGP implementations of log *P*. The CALCULATOR_INPUT_FORMAT table lists alternative input formats for each calculator; one of the listed formats must be made available to a calculator before it can start any work.

The table PROPERTY_DEPENDENCIES is used by the system to check if properties must be calculated in any particular order, because the calculation of some (derived) properties may require other (primary) properties as input. As an example, several estimation methods for log BB use simple expressions including log *P* and polar surface area (PSA), so running of those methods must wait until log *P* and PSA have been calculated and stored.

The dynamic part of the calculator dictionary is CALCULATOR_ACTIVITY, which is a list of connected calculators with their currently executing tasks, session identifiers, and status information. This table is updated by the functions of the task queue interface.

Job Definition Tables. Each job that is running or is scheduled to run has a record in the JOB table, see Figure 3. The column JOBSTART_FUNCTION stores the name of an Oracle procedure that will launch that type of job; those procedures must call the SUBMIT_TASKS_FOR_JOB function described below. The column JOBFINISHED_FUNCTION similarly lists any functions that are needed to postprocess the results from calculators. The JOB table also contains the scheduling priority as well as status and progress information about the job. The table JOB_PROPERTY_LIST lists all properties that needs to be calculated by a job, and the table JOB_DISPLAY_PROPERTY lists those properties requested by the user. If the job is a format conversion job, then JOB_FORMAT_LIST keeps records of what formats should be added by format conversion calculators. JOB_CALCULATOR_LIST lists the programs that will need to do work for the completion of a job. Records are inserted into these tables at job definition time by the procedures that code job types. The data in JOB_FORMAT_LIST and JOB_CALCULATOR_LIST are redundant in a strict sense because their content follow directly from the relations defined in the calculator dictionary. This organization is chosen anyway for reasons of modularity and clarity.

All jobs are started by the procedure STARTJOB, which takes a job identifier as an argument. This procedure issues an execute immediate to the job start function that was listed for the job. It is possible to specify that sequences of jobs are run in a particular order, and in such cases, the STARTJOB function will wait for completion of the first job before launching the next job.

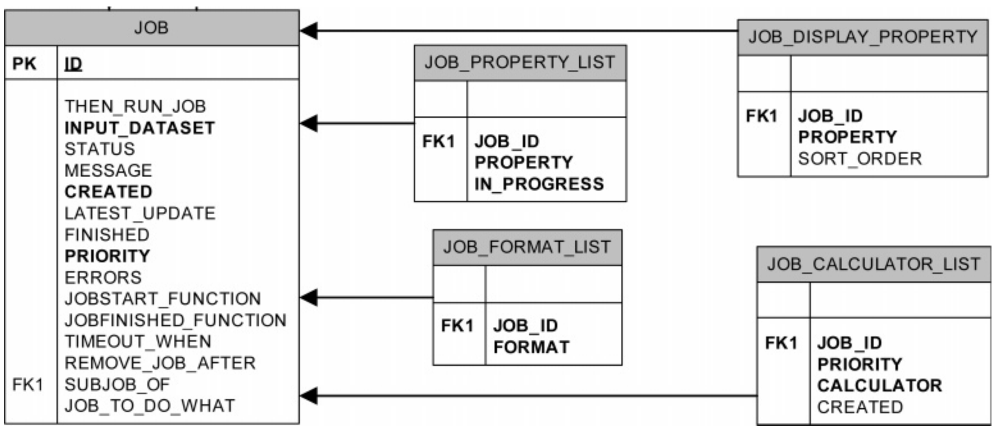

Figure 3. Job definition tables.

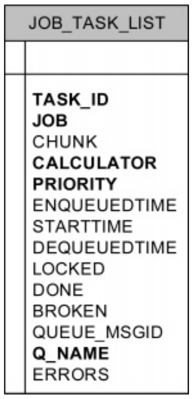

Figure 4. Task queue table.

Task Queue Table. The SUBMIT_TASKS_FOR_JOB function that must be called from all job start functions uses data in the calculator dictionary, the job definition tables, and the chunk table to enumerate the individual tasks that make up jobs. The tasks will be enqueued to an Oracle queue and put in the JOB_TASK_LIST table (Figure 4) with information about the chunk ID, the priority, and the status. Enqueueing a task effectively means that a calculator of specified type is invited to work on a particular chunk of the data in the input data set. The queuing system will assign one task to precisely one calculator instance.

Task Queue Interface. This interface is implemented in the TASKS package, which exposes the functions LOCK_TASK, DEQUEUE, SET_TASKFAILED, ENQUEUE_TASK, and GET_CHUNK_FOR_TASK. The LOCK_TASK function is at the heart of the system. A calculator calls this function when it is first started or becomes idle after having completed a task. Arguments to this function tell the task queue and the calculator dictionary the type of the calculator and whether the calculator deals with real-time jobs only. If one or more tasks of matching type are in the queue, then the identifier of the first enqueued task of highest priority will be returned. When assigned a task, a calculator is expected to go about with its work on the data repository, for example, calculating new properties. If the calculator completes the task successfully, it should call the DEQUEUE procedure, which commits the transaction and marks the task as done. If the calculator catches an error, then it should call SET_TASKFAILED with an error text argument; that procedure then marks the task as failed and logs the error.

Adaptors. An external program can be connected to the system by writing an adaptor program. A suitable program must expose some kind of automation interface or be controllable by command-line arguments. It is also required that output data be written to a file or have the ability to be sent over a standard database connection. The adaptor program will connect to the database, create a session, and call the LOCK_TASK procedure as described above. When a task identifier is returned in this call, the adaptor/calculator should take the following steps: (1) find the chunk to work on by calling GET_CHUNK_FOR_TASK; (2) find out what it needs to do by checking the job definition tables; (3) retrieve any data from the data repository that it needs for input; (4) start the external program with the input data, catch the output data, and write it to the data repository; (5) call DEQUEUE upon successful completion or, otherwise, SET_TASKERROR, preferably with an appropriate message; and (6) call LOCK_TASK to wait for the next task.

Calculator Example—Adaptor for Obabel. Pseudocode for a calculator calling the open-source format conversion program OpenBabel[14] is given in Figure 5. The figure outlines the steps; any real code is much longer.

User Interface. The application was developed for the primary purpose of serving an intranet site with calculated properties for both single structures and sets of structures. An interface for the Web application was written which consists of an Oracle package with functions for the import and export of chemical data and the launch of batch and single jobs and a function that waits for job completion (or timeout). Views with HTML-formatted results and error messages are also part of this interface.

This arrangement provides for enhanced modularity because there is a clear separation of Web server, presentation, and application logic. The Web server code is small and simple because most presentation logic is contained in the interface functions. An administrative interface was also written.

Security. Security is best enforced by an additional layer between the users and the application. If the application is driven by an intranet site, then the intranet application may be constructed to authenticate users and only send data in the sets owned by the user. This requires that the data set table be augmented with an owner column. A standard practice for achieving security is to grant no rights at all to users on the internal application objects but to construct

```
LOOP
  Begin
      Task_ID := TASKS.LOCK_TASK('BABEL',IsHighPrio);
      Chunk_ID := TASKS.GET_CHUNK_FOR_TASK(Task_ID);
      Get_requested_input_and_output_format();
      Export_structures_to_file(Chunk_ID,InFilename);
      varCMD := Build_commandline_statement_to_run_babel();
      Run_operating_system_command(varCMD);
      Add_babeloutput_to_database(Chunk_ID,OutFilename);

      TASKS.DEQUEUE (Task_ID);

      EXCEPTION WHEN OTHERS THEN
      TASKS.SET_TASKFAILED(Task_ID, ERRORMESSAGE);
  End;
END LOOP;
```

Figure 5. Pseudocode outlining the steps of an obabel adaptor module.

interface views and functions for users and grant appropriate access levels on those.

Calculators obviously must use accounts that have execute privileges on the task queue interface. They also need insert privileges on the property value tables and select privileges on the calculator dictionary, the job definition tables, and the task queue table.

RESULTS AND DISCUSSION

A key part of the application is the queuing system. The Oracle feature advanced queuing was used to implement the queue. In the application, the queue provides a mechanism to distribute tasks over many processes running in parallel on different CPUs. When a calculator has completed a task, it will wait at the queue for a new task in a blocking call which does not waste cycles on any of the machines involved.

Job and Task Distribution. Jobs with equal or different scheduling priority can run in parallel. The prioritization is done at the task level where tasks are dequeued in strict order of their parent job priority. The maximum number of simultaneously executing tasks is determined by the number of connected calculator instances. Load balance between connected machines handling tasks is maintained because machines with a heavy workload will request new tasks less often than those running few tasks.

Jobs are not transactional in the sense that their effect on a data set is undone when a job is deleted.

Fault Tolerance. The processes working with tasks are isolated from each other and will fail independently in case of errors. It is very unlikely that an error from a calculator propagates into the application processes. We deal with the following cases:

A. A calculator fails with an error, which it handles. The calculator calls the SET_TASK_ERROR procedure, which marks the task as failed, logs the error message, and issues a rollback to undo changes to the data repository. The task is taken away from the queue but can be re-enqueued from the job control API once the problem is found and fixed. It may be possible to do so without interfering with other jobs running. The application can be set up to retry the execution of tasks a predetermined number of times.

B. A calculator fails unexpectedly and dies. Oracle processes will detect the lost connection to the calculator and drop the corresponding database session. This will result in the automatic rollback of any changes made to the data. A procedure called CHECKSESSIONS, which runs periodically as an Oracle job, will switch the task status from locked to failed and update the calculator dictionary.

C. A calculator locks a task but does not complete it within the predetermined time. The CHECKSESSIONS procedure will issue a KILL SESSION IMMEDIATE on the session of the calculator and log the session as killed and the task as failed. The transaction of the terminated session is rolled back.

D. A network connection to a calculator drops. The result is same as that for B.

If the database instance itself fails and recovery is possible, then jobs will resume from the state at which recovery sets back the database. Jobs also resume after the database restarts, provided that the application is configured to re-execute uncompleted tasks.

A design goal was short runtimes and quick responses for high priority jobs, even at times when most of the computing capacity is used for long-running batch jobs. The tasks are dequeued in order of priority, but if all of the calculators are busy, then at least one of them must finish its current task and become idle before any new task can be handled. Keeping chunk sizes very small would provide one possible solution. A better way is to set up the system with some of the calculators dedicated to tasks of high priority. This is possible by calling the LOCK_TASK function from those calculators with the HIGH_PRIO argument set to True. When done so, only real-time tasks will be assigned to them.

Logging and Tracing. The application makes extensive use of logging at both the job and task levels. Job and task records persist until the jobs are deleted; error messages from calculators are stored until they are deleted by the administrator.

Performance. The import of drug-sized molecules from an SD file is done at a speed of about 10 000 molecules per minute. Import from a SMILES[15] file is on the order of 120 000 molecules per minute. The overhead from managing the job and task queues is on the order of 15 ms per task. A dummy calculator, which copies the structure representation to a buffer, calculates the character count, and writes this to the PROPERTY_VALUE table, works at a speed of about 2000 structures per second with SMILES input.

Platform and Vendor Dependence. Oracle is available from Oracle Corporation.[4] Oracle runs on all common operating systems, and database applications can easily be moved from one platform to another. The calculator adaptors can use any database connection API, such as ODBC (Open Database Connectivity) or JDBC (Java Database Connectivity), which means that database connectivity can be realized from most programming languages and operating systems.

PERSPECTIVES

The application was developed in response to extensibility problems that we experienced when new calculators were to be added to an existing intranet property predictor page. This was the original motivation, but very general usage within chemoinformatics should be possible. As an example, consider the extension to an automated QSAR system, which can store different established structure−activity models and automatically evaluate these using the appropriate calculators and properties, perhaps in response to the user input of structures from a Web page. Another possibility is the integration of a ligand−receptor docking step into an automated virtual screening workflow, with docking work distributed over many machines.

Supporting Information Available: Scripts for creation of the tables mentioned in the text and packages IO, JOBS, and TASKS. This information is available free of charge via the Internet at http://pubs.acs.org.

REFERENCES AND NOTES

(1) Scitegic. www.scitegic.com (accessed mmm yyyy).
(2) Stevenson, J. M.; Mulready, P. D. Pipeline Pilot 2.1 By Scitegic, 9665 Chesapeake Drive, Suite 401, San Diego, CA 92123-1365. www.scitegic.com. See Web Site for Pricing Information. *J. Am. Chem. Soc.* (Computer Software Review) **2003**, *125* (5), 1437−1438.
(3) Scitegic: What Is Data Pipelining? www.scitegic.com/products_services/what_is.htm (accessed mmm yyyy).
(4) Oracle Corporation. www.oracle.com (accessed mmm yyyy).
(5) Elsevier MDL. www.mdli.com (accessed mmm yyyy).
(6) Accelrys. www.accelrys.com (accessed mmm yyyy).
(7) Katritzky, A. R.; Petrukhin, R.; Tatham, D.; Denisenko, S. The Chemical Inventory System of the Center for Heterocyclic Compounds, University of Florida. *J. Chem. Inf. Comput. Sci.* **2002**, *42* (6), 1281−1282.
(8) Ben Miled, Z.; Liu, Y.; Powers, D.; Bukhres, O.; Bem, M.; Jones, R.; Oppelt, R. J. An Efficient Implementation of a Drug Candidate Database. *J. Chem. Inf. Comput. Sci.* **2003**, *43* (1), 25−35.
(9) Rojnuckarin, A.; Gschwend, D. A.; Rotstein, S. H.; Hartsough, D. S. ArQiologist: An Integrated Decision Support Tool for Lead Optimization. *J. Chem. Inf. Model.* **2005**, *45* (1), 2−9.
(10) Luque Ruiz, I.; Cerruela Garcia, G.; Gomez-Nieto, M. A. Clustering Chemical Databases Using Adaptable Projection Cells and MCS Similarity Values. *J. Chem. Inf. Model.* **2005**, *45* (5), 1178−1194.
(11) Fang, X.; Wang, S. A Web-Based 3D-Database Pharmacophore Searching Tool for Drug Discovery. *J. Chem. Inf. Comput. Sci.* **2002**, *42* (2), 192−198.
(12) Ertl, P.; Jacob, O. WWW-based Chemical Information System. *THEOCHEM* **1997**, *419*, 113−120.
(13) Tetko, I. V.; Tanchuk, V. Y.; Kasheva, T. N.; Villa, A. E. P. Internet Software for the Calculation of the Lipophilicity and Aqueous Solubility of Chemical Compounds. *J. Chem. Inf. Comput. Sci.* **2001**, *41* (2), 246−252.
(14) Main Page − Open Babel. http://openbabel.sourceforge.net (accessed mmm yyyy).
(15) Weininger, D. SMILES, a Chemical Language and Information System. *J. Chem. Inf. Comput. Sci.* **1988**, *28* (1), 31−36.

CI050360B