The Reaction Vessel: A General-Purpose Programming Framework Based on the Chemical Metaphor

Ben E. Cline
BenjysBrain.com
braininfo@benjysbrain.com

Abstract

The chemical metaphor, where some control aspects of a computational system are analogous to real chemistry, can provide a novel programming environment that provides features such as automatic concurrency and fault tolerance. It has been used in a number of systems based on such formalisms as the lambda calculus and rewrite rules in areas including distributed computing and network protocols. The Reaction Vessel extends the chemical metaphor to a general-purpose Java programming framework. To illustrate the features of the Reaction Vessel, a simple web crawler constructed from virtual molecules is described.

1. Introduction

Novel control mechanisms in programming environments provide elegant solutions to certain classes of problems and provide programmers with another idea that expands their "software development thought processes" [1]. The chemical metaphor[1], where computations are performed in a manner analogous to how chemical molecules interact, provides an interesting control mechanism that has been explored in a number of areas where the primary computation mechanism is based on a formalism such as multiset rewrite systems or the lambda calculus. The Reaction Vessel software described here expands the idea to a general-purpose Java programming framework that uses the chemical metaphor to provide a system that has an atypical control mechanism, provides automatic concurrency, is data-driven, includes stochastic control features, and provides a degree of fault tolerance.

There is an appeal to the idea of putting virtual data and program molecules in a virtual reaction vessel that is well-stirred and having the program molecules collide with and process the data molecules whose structure they were designed to recognize. Imagine a virtual reaction vessel containing a soup of web pages, for example, and program molecules that extract particular features, such as images, from the pages. Other program molecules might then examine the extracted images and discard those that don't meet a certain criterion while retaining the qualified images, such as those that meet a particular size requirement. By casting the web crawler problem in terms of a set of chemical reactions, the design of the web crawler software can take advantage of automatic concurrency and fault tolerance that is part of the chemical metaphor control mechanism.

In the general case, the result of the collision between a program and a data molecule can be empty or a set of one or more new data molecules including a regenerated or modified copy of the input data molecule. A sequence of operations can occur when a number of program molecules performing computations pass modified data molecules into the environment containing the program molecules, providing a mechanism for problem decomposition by having parts of the problem handled by different program molecules. Because each collision involves a single program molecule and a copy of a data molecule, all data is local to the collision and does not affect other collisions. Therefore, collisions can be processed in different threads and even in a distributed manner. Concurrency is automatic at the collision level, and the programmer handles higher level synchronization through the production of data molecules in a chain of operations.

A general-purpose programming environment with a chemical metaphor control mechanism can operate in a deterministic mode where the order of molecular collisions is predictable and all possible collisions occur and a stochastic mode where the order of collisions is pseudo-random and where only a subset of possible collisions occur. The latter more closely models real chemistry and can be useful in dealing with problems which do not have a single correct answer or for intractable problems where a good problem solution is adequate. Due to the concurrency, non-deterministic

[1] The chemical metaphor is sometimes called "chemical programming" or the "chemical reaction metaphor".

results can also occur in some problems where race conditions affect the outcome of the computation at hand.

Systems based on the chemical metaphor can be formalized in terms of an artificial chemistry which makes the explanation of a particular system clearer. In a later section, a definition of the Reaction Vessel is given in terms of an artificial chemistry. Section 4 describes an example of a web crawler implemented in the Reaction Vessel environment.

2. Background

The chemical metaphor has been used in a number novel computation systems. Gamma and extensions are based on multisets and rewrite rules [2][3][4], while Andrei and Kirchner [5] use a port graph rewriting system. Other systems are based on the lambda calculus [6] or production rules as in the Chemical Casting Model (CCM) [7].

Chemical metaphor-based systems have been proposed for a variety of problem areas including machine instruction ordering [8], scientific workflow management in a distributed environment [9], computer networking [10][11], and service-oriented architectures [12]. Lin and Yang [13] have applied the chemical metaphor to multi-agent systems, and Banatre *et al* [14] have applied the chemical metaphor to grid programming.

In the field of artificial life, artificial chemistries [15] have been created to explore such concepts as evolution, membrane formation, and self organization; however, the artificial chemistry formalism is useful for describing chemical metaphor systems used in both artificial life and computational frameworks [16]. Artificial chemistry systems can be divided into two broad subclasses: ones that simulate individual collisions and those that use differential or difference equations to simulate rates of change of molecular concentrations. For systems that perform general-purpose computations, individual collisions are simulated.

In contrast to the systems based on formalisms such as rewrite rules, the Reaction Vessel is a general-purpose programming framework. Although the system is defined as an artificial chemistry, individual program molecules are Java classes with a limited number of restrictions. While providing many of the benefits of other chemical metaphor-based systems, the Reaction Vessel program molecules can take advantage of the object-oriented paradigm and a wide variety of Java packages including XML/XHTML parsers, the XPath XML node extraction package, and a regular expression subsystem.

3. The Reaction Vessel

The Reaction Vessel is first described in terms of an artificial chemistry, and then details are given about how the Reaction Vessel is used to solve computational problems.

3.1 Reaction Vessel as an Artificial Chemistry

Informally, an artificial chemistry is a man-made chemistry [15]. Although an artificial chemistry definition can apply to a physical system, many artificial chemistries are software simulators. In the field of artificial life, artificial chemistries are typically used to explore the mechanisms behind biological systems; however, it is sometimes useful to cast a computational system as an artificial chemistry.

Systems based on the chemical metaphor can be formalized in terms of an artificial chemistry. An artificial chemistry is defined by a triple (S, R, A), where S is the set of all possible molecules in the system, R is a set of collision rules, and A is an algorithm describing a virtual reaction vessel in which molecules collide [15]. In the Reaction Vessel, S is the set of program molecules and data molecules that are possible in the system. In some configurations, the set of program molecules is fixed while the set of data molecules contains the initial set of seed data molecules and those produced by program molecules. But there is no restriction on adding and removing program molecules as the Reaction Vessel executes.

In Reaction Vessel, there is one reaction rule in R:

$$PM_i + DM_j \rightarrow \varepsilon \mid \{DM_k, DM_{k+1}, ..., DM_{k+m}\}$$

where PM_i is the i^{th} program molecule, DM_j is the j^{th} data molecule. The output of the collision between two molecules is zero or more data molecules labeled DM_k to DM_{k+m}. Each program molecule operates on a copy of DM_j, and the data molecule is then removed from the system unless a copy is regenerated by any of the program molecules. Program Molecule PM_i is not removed from the system; however, the thread running a collision is terminated when that collision is complete. By default, all data molecules are processed by each program molecule; however, if the data molecule is not one that the program molecule recognizes, no further action is taken. A program molecule can implement a concentration method that is based on the idea of the concentration of a substance in a real chemical reaction vessel. Based on a pseudo-random distribution, a program molecule can

randomly operate on a subset of the data molecules with which it is presented. Decisions to operate on particular data molecules can also be made based on measures of system dynamics.

Data molecules are stored in a priority queue. Those with the highest priority are processed first. The priority scheme is typically used to allow control data molecules produced by program molecules to be processed by the reaction vessel software immediately. Control data molecules are not processed by other program molecules but are messages processed directly by the reaction vessel software.

In the artificial chemistry formalism, *A* defines the reaction vessel algorithm. In the Reaction Vessel, it implements the reaction rule in the following manner:

- If the next data molecule in the priority queue is a control data molecule, it is processed immediately. Such messages from program molecules can cause a data molecule to be queued for later processing, limit the number of concurrent threads, or indicate a termination condition.
- If the data molecule is not a control data molecule and the programmer-defined thread limit has not been exceeded, each program molecule is queried to determine if the data molecule is of the type it is designed to process. For each compatible program molecule/data molecule pair, a thread is started for the program molecule with a shallow copy of the data molecule. If the thread limit is exceeded, the control loop waits until enough threads complete before processing the data molecule.
- When a program molecule is activated, it executes the concentration method. If the concentration decision is false, the program molecule thread terminates. Otherwise, the data molecule is processed, and zero or more data molecules are added to the data molecule priority queue.

Figure 1 illustrates the Reaction Vessel control mechanism.

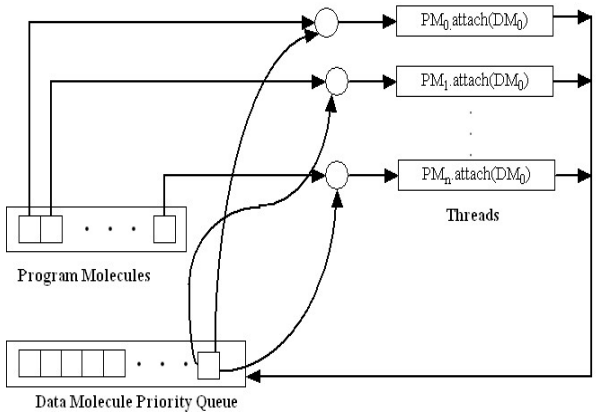

Figure 1 - The Reaction Vessel processing of the highest priority data molecule DM_0 by creating threads to run *attach(DM_0)* for each compatible program molecule PM_0 to PM_n.

It should be noted that the Reaction Vessel is trivially Turing complete. A universal Turing machine can be implemented as a program molecule, and the Turing machine tape can be implemented as a single data molecule. The Reaction Vessel is seeded with a single data molecule representing the initial tape configuration. At each step of the Reaction Vessel, the Turing machine program molecule would consume the data molecule representing the tape, do one Turing machine step, and produce a single data molecule representing the new tape configuration. When the program molecule reaches its final step, it could output a special data molecule intended to display the contents of the final tape and to halt execution of the Reaction Vessel.

3.2 Programming with the Reaction Vessel

The Reaction Vessel is a Java implementation of the artificial chemistry defined in the previous section. There are three primary object classes in the system: ReactionVessel, ProgramMolecule, and DataMolecule. The latter two classes are abstract.

The Reaction Vessel (class ReactionVessel) implements the chemical metaphor control mechanism. It contains a list that holds program molecules (subclasses of class ProgramMolecule) and a priority queue that holds data molecules (subclasses of class DataMolecule). To create a program, the programmer instantiates a ReactionVessel object, adds program molecule objects, and seeds the system with one or more data molecules. The Reaction Vessel then runs the program by implementing the algorithm described in the previous section. The system runs until there are no data molecules remaining in the system, after a fixed number of

collisions, or when a control data molecule that indicates a stopping condition is sent to the Reaction Vessel .

There are no restrictions on the implementation of ProgramMolecule classes other than a requirement for thread safety and that they extend the ProgramMolecule class. Data molecules are implemented by classes that extend the DataMolecule class. The only restrictions on DataMolecule subclasses are that they provide a shallow copy method and a "getter" method to access the data.

The Reaction Vessel includes two classes for dealing with formatted data: a data molecule that holds field data and a regular expression processing program molecule. The field data molecule holds a tree of data fields. Fields are text, numeric, or other data types. A regular expression program molecule attaches to a field data molecule, applies the regular expression pattern with which it was instantiated, and if the pattern matches, reformats the field tree. One use of these classes is to recognize standard data formats such as SQL insert statements and comma-separated records, and to convert the data into another format. With a library of regular expression molecules with patterns that recognize certain data formats and extract fields from them, exemplar data could be placed in the reaction vessel, and the regular expression molecules would recognize the format and provide a template for dealing with large data sets. The regular expression program molecule also supports string matching and substitution on a field tree.

Although program molecules can only operate on a single data molecule at a time, a special list data molecule can hold references to a number of data molecules. To create a data molecule for a specific program molecule, other helper program molecules would capture, preprocess, and merge the needed data molecules into a list data molecule.

Although program molecules cannot collide with each other, they can still exchange information via data molecules. Using this method, communicating agents can be implemented as program molecules in the Reaction Vessel.

The Reaction Vessel contains several mechanisms to control the reaction rate of the computation at hand. First, the programmer provides a thread limit, as described above. Second, the thread limit can be modified dynamically by a program molecule by generating a control data molecule and adding it to the data molecule priority queue. Third, a program molecule that is consuming a large amount of computational resources in a number of threads can requeue the data molecule it has been given and process it when resources become available. These mechanisms provide a means to limit the number of concurrent threads and prevent resource thrashing.

4. Demonstration

A simple web crawler was implemented using the Reaction Vessel. The crawler selects web images that fit an empirically derived formula [17] that defines the typical size and aspect ratio of newspaper-style comics that are found on the web. The following program molecules are added to the virtual reaction vessel to implement the web crawler:

- Fetcher – This program molecule processes a data molecule containing a URL and uses an HTML parser to create a data molecule containing a Document Object Model (DOM) that represents the web page.
- Harvester – This program molecule uses XPath to retrieve URLs from a DOM. There are two instances of this class in the web crawler. Harvester I extracts hyperlink URLs, while Harvester II extracts image URLs. The former are placed in data molecules recognized by the Fetcher program molecule, while the latter are placed in data molecules intended for the ImageFetcher program molecule.
- ImageFetcher – This program molecule fetches an image based on a URL and saves it to a database if it matches the size and aspect ratio of a comic.

The system uses two types of data molecules:

- A URL data molecule with two fields. One contains a URL, and the other contains information about the URL, e.g., if it is a web page URL or an image URL.
- A data molecule that holds the DOM of a parsed page.

Figure 2 is a diagram of the web crawler system. The Reaction Vessel is started with one instance of the Fetcher program molecule, two instances of the Harvester program molecule (one for processing hyperlinks and one for processing images), and an instance of the ImageFetcher program molecule. It is also seeded with the URL of one of more web pages containing comics or links to comic pages.

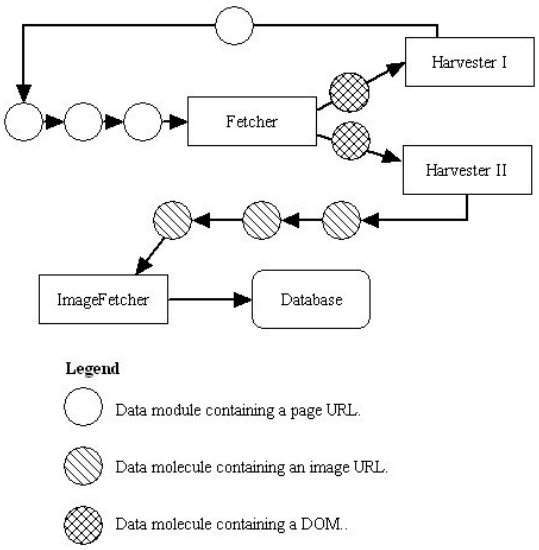

Figure 2 – Reaction Vessel web crawler

As the system runs, the Fetcher loads and parses web pages, the Harvesters extract web page and image URLs, and the ImageFetcher loads and tests images for potential comics. The Harvester that extracts web page URLs produces work for the Fetcher.

This web crawler illustrates the need for reaction control. Because of the feedback loop between Harvester I and the Fetcher, it is easy for the system to go into a state where thread and memory resources are exhausted. Simple tuning can be performed by setting a thread limit. The Fetcher is also designed to limit the number of threads that are concurrently executing its *attach()* method when there are too many threads fetching web pages by sending a control data molecule to the Reaction Vessel to requeue its current data molecule for later processing.

One problem with implementing the web crawler in the Reaction Vessel is that a collision between a data molecule and a program molecule requires global data to execute efficiently. For example, the Fetcher program molecule might attach to a data molecule referencing a web page it has already fetched. Fetching the page again is wasteful and unnecessary. In some cases such as this, a static instance variable in the Fetcher object can hold a hash table of previously fetched pages. In other cases, the creation of high-priority data molecules containing control information can be passed between different program molecules to implement this type of information sharing.

The fault tolerant aspects of the Reaction Vessel are useful in the simple web crawler. Because of the thread architecture, a thread that is running the *attach()* method of a program molecule can die with an exception without affecting other threads executing the same and other program molecules. Optionally, information from uncaught threads can be turned into data molecules that are processed by a user-supplied program molecule. This mechanism allows the programmer to design a program molecule that exerts high-level control if it is needed for the problem at hand.

5. System Evaluation

The Reaction Vessel framework is intended to relieve the programmer of the task of low-level thread creation and synchronization and to provide a means of problem decomposition using the chemical metaphor. To perform these tasks, the Reaction Vessel has a control loop thread that monitors the data molecule queue and starts threads for program molecule/data molecule pairs as the thread limit allows. Reaction Vessel overhead comes from this loop, the implementation of the Java Object *wait()* and *notify()* thread synchronization methods, the creation of new threads, and the creation of shallow copies of the data molecules. Each time a program molecule/data molecule pair is selected, the Reaction Vessel runs a method in the program molecule to determine if it is compatible with the data molecule. If it is, it creates a shallow copy of the data molecule and starts a new thread in the existing program molecule object.

Additional delays occur when data molecules in the priority queue are blocked from being processed because of the thread limit. The result is that some program molecules might have many threads executing their *attach()* method while other program molecules are dormant. In some cases, this situation might be intended; however, in other cases the priority queue might become a throughput bottleneck. Careful use of priority queue priorities can reduce the delay in some cases.

Simple performance testing was preformed between the Reaction Vessel and a hand-coded test program. In one case where one or more Fast Fourier Transforms (FFT) were used as the load for a single thread, the means of the sampled computation times for the Reaction Vessel and the hand-coded program for various numbers of threads was not significantly different. A similar test where the program molecule calculation was trivial showed that the Reaction Vessel was significantly slower than the hand-coded test program. The reduction in performance was not due to a priority queue bottleneck in this case but to the control loop overhead.

When considering the Reaction Vessel as a tool to solve a particular problem, one should consider whether

the chemical metaphor is suited to the problem and how the control loop and priority queue bottlenecks will impact performance.

6. Conclusion and Future Work

By casting the Reaction Vessel as an artificial chemistry, a precise statement of Reaction Vessel semantics is possible. The resulting framework is small, easy to understand, and easily extendable. Programmers can create new programs in the framework by either instantiating existing program molecules or by extending existing ones.

The Reaction Vessel has features of the chemical metaphor including automatic concurrency and provides a different way to think about general-purpose computing. The system can be used to solve complex programming problems as illustrated by the simple web crawler in a way that helps the developer frame the problem in a novel way.

The Reaction Vessel has primarily been applied to problems related to parsing web pages and the automatic recognition of data structures and data reformatting; however, it can be used for numeric computations and other problem areas where concurrency is desirable. A larger library of program molecules that would make the system more easily extendable to other types of problems is planned.

One of the benefits of using a system with the chemical metaphor is the automatic concurrency at the collision level; however, the thread-based concurrency of the current version is limited. A distributed version would be more useful; however, concurrent access to program molecule instance variables between distributed instances of the class would not be possible. Other control mechanisms based on the passing of data molecules would be necessary.

7. References

[1] R. W. Sebesta, *Concepts of Programming Languages*, Seventh Edition, Addison-Wesley, Reading, Massachusetts, 2006.

[2] J-.P. Banatre and D. le Metayer, “A new computational model and its discipline of programming”, Technical report no. 566, INRIA, 1986.

[3] G. Berry and G. Boudol, “The chemical abstract machine”, Theoret. Comput. Sci. 96 (1992) 217–248.

[4] G. A. L. Paillard, F. M. G. Franca, K. A. M. Filho, "A distributed implementation of structured gamma," p. 0445, Eighth International Conference on Parallel and Distributed Systems (ICPADS'01), 2001.

[5] O. Andrei and H. Kirchner, “Strategic port graph rewriting for autonomic computing”. In: Proc. of TFIT'08. (2008).

[6] Fontana, W., “Algorithmic chemistry”, In C. G. Langton, C. Taylor, J. D. Farmer, & S. Rasmussen (Eds.), *Artificial Life II* (pp. 159–210). Redwood City, CA: Addison-Wesley, 1992.

[7] Y. Kanada, and M. Hirokawa, “Stochastic problem solving by local computation based on self-organization paradigm”, *IEEE 27th Hawaii International Conference on System Sciences*, pp. 82-91, 1994.

[8] W. Banzhaf and C. W. G. Lasarczyk, “A new programming paradigm inspired by artificial chemistries”, In J.-P. Banˆatre, J.-L. Giavitto, P. Fradet, and O. Michel, editors, *Unconventional Programming Paradigms* (UPP–04), LNCS, Berlin, 2005. Springer.

[9] Nemeth, Z.; Perez, C.; Priol, T., "Workflow enactment based on a chemical metaphor," *Software Engineering and Formal Methods*, 2005. SEFM 2005. Third IEEE International Conference on, vol., no., pp. 127-136, 7-9 Sept. 2005.

[10] T. Meyer, L. Yamamoto, and C. Tschudin, “An artificial chemistry for networking”, *Bio-Inspired Computing and Communication*, 2008, pp. 45–57.

[11] Meyer, T., Tschudin, C., “Chemical networking protocols”, In: *Proc. 8th ACM Workshop on Hot Topics in Networks* (HotNets-VIII). (2009).

[12] J.-P. Banatre and T. Priol. “Chemical programming of future service-oriented architectures”, *Journal of Software*, 4(7) p. 738-746, 2009.

[13] H. Lin and C. Yang, “Specifying distributed multi-agent systems in chemical reaction metaphor”, *The International Journal of Artificial Intelligence, Neural Networks, and Complex Problem Solving Technologies*, Springer-Verlag, 24(2), 2006, pp. 155-168.

[14] Banatre, J.P., Le Scouarnec, N., Priol, T., Radenac, Y.: “Towards ‘chemical’ desktop grids”, In*: Proceedings of the 3rd IEEE International Conference on e-Science and Grid Computing* (e-Science 2007), IEEE Computer Society Press, Los Alamitos (2007).

[15] P. Dittrich, J. Ziegler, W. Banzhaf, “Artificial chemistries: A review”, *Artificial Life* **7**, 225–275 (2001).

[16] P. Dittrich, “Chemical computing”, In Jean-Pierre Banâtre, Pascal Fradet, Jean-Louis Giavitto, and Olivier Michel, editors, *UPP*, volume 3566 of *Lecture Notes in Computer Science*, pages 19–32. Springer, 2004.

[17] B. Cline, “Automatic extraction of comics from web pages”, <http://www.benjysbrain.com/misc/comics/>, 2009.