Towards Grid Chemical Coordination (short paper)

Jean-Pierre Banâtre, Pascal Fradet, Yann Radenac

Towards Chemical Coordination for Grids

Jean-Pierre Banâtre[1], Pascal Fradet[2] and Yann Radenac[1]
[1] IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France
[2] INRIA Rhône-Alpes, 655 avenue de l'Europe, 38330 Montbonnot, France

In [6], Ian Foster and Karl Kesselman explain that grids need "a rethinking of existing programming models and, most likely, new thinking about novel models". In this work, we investigate a "novel programming model" for grids based on the chemical metaphor.

On one side, applications are programmed in an abstract manner describing essentially the chemical coordination between (not necessarily chemical) software components. On the other side, chemical service programs are specifically provided to the Grid run-time system in order to obtain the expected quality of service in terms of efficiency, reliability, security, etc. These service programs can be seen as special coordination programs providing guidelines to the runtime system allowing a better use of resources in order to obtain the expected Quality of Service.

1. CHEMICAL PROGRAMMING

A chemical program can be seen as a (symbolic) chemical solution where data is represented by floating molecules and computation by chemical reactions between them. When some molecules match and fulfill a reaction condition, they are replaced by the body of the reaction. That process goes on until an inert solution is reached: the solution is said to be inert when no reaction can occur anymore. Formally, a chemical solution is represented by a multiset and reaction rules specify multiset rewritings.

We use a higher-order chemical programming language called HOCL(Higher-Order Chemical Language). HOCL [2] is based on the γ-calculus [3], a higher-order chemical computation model which can be seen as an higher-order extension of the Gamma language [4]. In HOCL, every entity is a molecule, including reaction rules.

A program is a molecule, that is to say, a multiset of atoms $(A_1, \ldots, A_n)$ which can be constants (integers, booleans, etc.), sub-solutions (denoted by $\langle M \rangle$), pairs (denoted by $A_1{:}A_2$) or reaction rules (denoted by **replace-one** P **by** M **if** C, where P is a pattern which filters the required molecule, C is the reaction condition and M the result of the reaction). In particular, the pattern $\langle P \rangle$ matches an inert sub-solution, and the pattern $x{::}T$ is used to filter atoms of a given type T.

Compound molecules (M_1, M_2) are built using the associative and commutative operator ",":

$$(M_1, M_2), M_3 = M_1, (M_2, M_3) \qquad M_1, M_2 = M_2, M_1$$

formalize the Brownian motion and can always be used to reorganize molecules.

The execution of a chemical program, i. e., a chemical solution, consists in triggering reactions until the solution becomes inert.

A reaction involves a reaction rule **replace-one** P **by** M **if** C and a molecule N that satisfies the pattern P and the reaction condition C. The reaction consumes the rule, the molecule N and produces M.

A molecule inside a solution cannot react with a molecule outside the solution (the construct $\langle . \rangle$ can be seen as a membrane). A HOCL program is a solution which can contain reaction rules that manipulate other molecules (reaction rules, sub-solutions, etc.) of the solution.

The reaction rules are one-shot: they are consumed when they react. However, in many programs it is practical to have n-shot reaction rules (like in Gamma [4]) i. e., rules which do not disappear when they react. Like in Gamma, there are denoted by **replace** P **by** M **if** C.

There are usually many possible reactions making the execution of chemical programs non-deterministic and highly parallel. Since reactions involves only a few molecules and react independently of the context, many distinct reactions can occur at the same time. For example, consider the following program that computes the prime numbers lower than 10 using a chemical version of the Eratosthenes' sieve:

$$prime10 = \textbf{let } sieve = \textbf{replace } x, y \textbf{ by } x \textbf{ if } x \text{ div } y \textbf{ in} \\ \langle sieve, 2, 3, 4, 5, 6, 7, 8, 9, 10 \rangle$$

The reaction *sieve* just removes any element y which can be divided by another one x. Initially several reactions are possible, for example $sieve, 2, 8$ (replaced by $sieve, 2$) or $sieve, 3, 9$ (replaced by $sieve, 3$) or $sieve, 2, 10$ or etc. The solution becomes inert when *sieve* cannot react with any couple of integers in the solution, that is to say, when the solution contains only prime numbers. The result of the computation in our example is $\langle sieve, 2, 3, 5, 7 \rangle$.

To access within a sub-solution (e. g., to get the result of a sub-program), a reaction rule has to wait for its inertia. That means that a reaction rule matches only inert sub-solutions. For example, if we want to compute the largest prime number lower than 10, we can use the previous program as a sub-program and then computes the maximum of

SAC'06 April 23-27, 2006, Dijon, France

its result:

$$\textbf{let}\ max = \textbf{replace}\ x, y\ \textbf{by}\ x\ \textbf{if}\ x \geq y\ \textbf{in}$$
$$\langle prime10,\ \ \textbf{replace-one}\ \langle \omega \rangle\ \textbf{by}\ \omega, max \rangle$$

When the solution $prime10$ becomes inert, the result is extracted and the reaction max computes the maximum prime number.

2. CHEMICAL COORDINATION AND GRID PROGRAMMING

HOCL is a general coordination language: it can combine programs and ensure QoS properties on their execution. Grids also can be represented as solutions of resources. It is then easy to specify tasks placement or migration using HOCL. The resulting system possesses nice autonomic properties as shown in [1].

Thanks to its higher-order nature, it is easy to define coordination mechanisms in HOCL. For example, the parallel operator is simply the comma: if M_1 and M_2 are two tasks, the molecule $(\langle M_1 \rangle, \langle M_2 \rangle)$ represents their parallel execution.

HOCL can be seen as a chemical coordination language where the coordinated objects are data and programs. Programs are considered as black boxes which take some (typed) arguments and yield a (typed) result (for example they may be defined in an external library). For example, our previous example *sieve* could be seen as the coordination of the function *div* on integers in order to compute the prime numbers.

A grid is represented by a solution of resources (e. g., all machines in the grid). Resources have characteristics accessible through predefined functions. For example, $Cpu(R)$ returns the type of cpu of a resource R, $Load(R)$ yields its current load, $MemSize(R)$ returns the size of its available memory, etc.

A program to be run on a grid is a set of tasks distributed on several resources. A grid is a dynamic system: resources come and go, some of them become overloaded or available, etc. So, before launching a program into a grid, it is in general impossible to know what resources will be allocated to the program. The coordinator can specify the dynamic placement or the migration of tasks using reaction rules. For example, the programmer may specify that a task allocated to a resource may be migrated to a less loaded resource.

Initially, tasks are placed on some free resources and reaction rules expressing the coordination (migration, duplication, etc.) of tasks are placed among the solution of resources. A resource is represented as a tagged solution $R{:}\langle x, \ldots, f{:}a, \ldots \rangle$ where the solution contains the active and idle tasks placed on the resource named (tagged) R. An idle task is represented by a function-argument pair $(f{:}a)$. An active task is an expression (fa) or any active solution. For example, the activation of idle tasks on a resource can be specified by the reaction rule:

$$activate = \textbf{replace}\ r{:}\langle \omega, (f{:}a) \rangle \ \textbf{by}\ r{:}\langle \omega, fa \rangle \ \textbf{if}\ Load(r) < 80\%$$

Placed in a grid, this rule will activate any idle task on any resource whose load is less than 80%. The migration of idle tasks on a less loaded resource can be expressed by the reaction rule:

$$migrate = \textbf{replace}\ r_1{:}\langle \omega_1, (f{:}a) \rangle,\ r_2{:}\langle \omega_2 \rangle \ \textbf{by}\ r_1{:}\langle \omega_1 \rangle,\ r_2{:}\langle \omega_2, (f{:}a) \rangle \ \textbf{if}\ Load(r_2) < Load(r_1) - 10\%$$

A basic load balancing can be specified by placing these two rules in the solution representing the grid. For example

$$\langle R_1, R_2, R_3, \ldots, R_n, activate, migrate \rangle$$

In general, coordination on a grid is specified by many such rules for load balancing but also fault tolerance, security, etc. These reactions can take place in parallel and leads to non deterministic task placement or migration. Execution proceeds until self-stabilization that is, until the solution becomes inert.

3. CONCLUDING REMARKS

We have presented a chemically inspired approach that unifies grid programming and coordination.

Concerning related work, let us mention two approaches which bare some similarities to our approach. In [7], it is shown how a rule-based multiset programming paradigm close to Gamma [4] may be used as a unifying theme for various models of computation, such as biological , molecular, DNA, etc. The Organic Grid [5] is another effort based on a biologically inspired paradigm.

Practical realization of our approach is presently underway on an experimental grid architecture. We are developing a chemical grid virtual machine built from interconnected chemical processing elements providing the basic functionalities expected for program execution and coordination.

4. REFERENCES

[1] J.-P. Banâtre, P. Fradet, and Y. Radenac. Chemical specification of autonomic systems. In *Proc. of the 13th Int. Conf. on Intelligent and Adaptive Systems and Software Engineering (IASSE'04)*, 2004.

[2] J.-P. Banâtre, P. Fradet, and Y. Radenac. Generalized multiset for chemical programming. Technical report, INRIA, 2005. (to appear).

[3] J.-P. Banâtre, P. Fradet, and Y. Radenac. Principles of chemical programming. In S. Abdennadher and C. Ringeissen, editors, *Proceedings of the 5th International Workshop on Rule-Based Programming (RULE 2004)*, volume 124 of *ENTCS*, pages 133–147. Elsevier, 2005.

[4] J.-P. Banâtre and D. Le Métayer. Programming by multiset transformation. *Communications of the ACM (CACM)*, 36(1):98–111, Jan. 1993.

[5] A. J. Chakravarti, G. Baumgartner, and M. Lauria. Application-specific scheduling for the organic grid. In *Proceedings of the fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*, pages 146–155, 2004.

[6] I. Foster and C. Kesselman, editors. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, 2nd edition, 2003.

[7] V. K. Murthy and E. V. Krishnamurthy. Gamma programming paradigm and heterogeneous computing. In *Proceedings of the 29th Hawaii International Conference on System Sciences (HICSS'96) Software Technology and Architecture*, volume 1, page 273, 1996.