**Journal of Simulation (2013) 7**, 202–215 © 2013 Operational Research Society Ltd. All rights reserved. 1747-7778/13

www.palgrave-journals.com/jos/

# Chemical-oriented simulation of computational systems with ALCHEMIST

D Pianini, S Montagna and M Viroli*

*DISI-Universita di Bologna, Cesena, Italy*

In this paper we address the engineering of complex and emerging computational systems featuring situatedness, adaptivity and self-organisation, like pervasive computing applications in which humans and devices, dipped in a very mobile environment, opportunistically interact to provide and exploit information services. We adopt a meta-model in which possibly mobile, interconnected and communicating agents work according to a set of chemical-like laws. According to this view, substantiated by recent research on pervasive computing systems, we present the Alchemist simulation framework, which retains the performance of known Stochastic Simulation Algorithms for (bio)chemistry, though it is tailored to the specific features of complex and situated computational systems.

*Journal of Simulation* (2013) **7,** 202–215. doi:10.1057/jos.2012.27; published online 18 January 2013

**Keywords:** distribution; pervasive systems; simulation; bio-inspiration

## 1. Introduction

Complexity is everywhere, both in nature—whether we observe it at the physical, chemical, biological or social level (Zambonelli and Viroli, 2011)—and in artificial computing systems, realised to support nowadays scenarios like those of pervasive computing, which leverage the increasing large-scale availability of smart computational devices. Such natural and artificial systems seem to share a common set of properties including situatedness, adaptivity and self-organisation, which altogether make them react to unpredictable situations with no explicit design-time or supervised control, namely, by emergence. It comes not as a surprise, then, that the design of mechanisms and algorithms to be adopted in emerging computing systems typically rely on the two following key pillars.

On the one hand, they leverage natural inspiration, either in terms of patterns (Gardelli *et al*, 2007; Fernandez-Marquez *et al*, 2012) or of metaphors (Babaoglu *et al*, 2006; Zambonelli and Viroli, 2011). A notable class of such approaches, which we focus on in this paper, falls under the umbrella of chemical computing. This research thread originated in the work of Gamma (Banâtre and Métayer, 1993; Banâtre *et al*, 2001) and the chemical abstract machine (Berry and Boudol, 1992), then developed in a plethora of theoretical models of computing (see, eg, Bergstra and Bethke, 2002; Credi *et al*, 2007; Brijder *et al*, 2011), it was extended with a structuring of the environment in compartments as in biological systems (like in Ambients Cardelli and Gordon, 2000 and P-systems Paun, 2002), and recently it has been framed in the context of coordination models (Viroli and Casadei, 2009) with applications to the field of pervasive computing (Viroli and Zambonelli, 2010; Viroli *et al*, 2011; Zambonelli *et al*, 2011).

On the other hand, the development methodology for such systems always include *simulation* as a key step (Babulak and Wang, 2008; Bandini *et al*, 2009a; Macal and North, 2010; Molesini *et al*, 2011) to realise what-if analysis prior to actual development, to both assess the general validity of the designed mechanisms and to fine tune system parameters. There are many kinds of simulation tools available: they either provide programming/specification languages devoted to ease construction of the simulation process, especially targeting computing and social simulation (eg, as in the case of multi-agent based simulation (North *et al*, 2007; Schumacher *et al*, 2007; Sklar, 2007; Bandini *et al*, 2009a; Bandini *et al*, 2009b), or they stick to quite foundational computing languages to better tackle performance, mostly used in biology-oriented applications (Murata, 1989; Priami, 1995; Uhrmacher and Priami, 2005; Ewald *et al*, 2007).

The ALCHEMIST simulator presented in this paper is aimed at bridging the gap between these approaches, targeting chemical-oriented computational systems. It extends the basic computing model of chemical reactions—still retaining its high performance—toward ease applicability to complex situated computational systems (following the chemical-oriented abstractions studied in Zambonelli *et al*, 2011). In particular, ALCHEMIST is based on an optimised version of the Gillespie's SSA (Gillespie, 1977) called Next Reaction Method (Gibson and Bruck, 2000), properly extended with

*Correspondence: M Viroli, DISI-Universita di Bologna, Alma Mater Studiorum Universita di Bologna, via Venezia 52, Cesena, FC 47521, Italy.*
E-mail: mirko.viroli@unibo.it

the possibility to deal with a mobile and dynamic 'environment' (adding/removing reactions, data-items and topological connections). The underlying meta-model and simulation framework are built to be pretty generic, and as such they can have a wide range of applications beside pervasive computing, including social interactions and computational biology (Montagna *et al*, 2012).

To exemplify the approach we illustrate a case study of crowd steering in pervasive computing, in which groups are guided towards locations based on their preference, along optimal paths and taking into account the presence of crowded regions that should be circumvented. We provide the set of reactions that model behaviour of agents (human, devices, sensors) and environment behaviour, adopting self-organisation mechanisms proposed in the context of spatial computing (Mamei and Zambonelli, 2009; Viroli *et al*, 2011) and validate them via simulation of the associated stochastic model.

The remainder of this paper is organised as follows: Section 2 introduces and motivates the chemical meta-model we rely upon, Section 3 presents the simulation engine (its computational model and inner architecture), Section 4 reports on its application to the crowd steering scenario and the corresponding simulation results, Section 5 compares (from both a qualitative and quantitative way) our work with existing agent-based models (ABMs) and simulation frameworks and Section 6 provides concluding remarks and discusses future works.

## 2. On the chemical-oriented meta-model

The complexity of the systems we want to face in this paper is achieved by the following set of common key properties:

- Situatedness—they deal with spatially and possibly socially situated activities of entities, and should therefore be able to interact with a limited portion of the surrounding world and contextualise their behaviour accordingly.
- Adaptivity—they should inherently exhibit properties of autonomous adaptation and management to survive contingencies without external intervention, global supervision, or both.
- Self-organisation—spatial and temporal patterns of behaviour should emerge out of local interactions and without a central authority that imposes pre-defined plans.

Among the many natural metaphors one can use as inspiration for modelling and developing artificial systems with the above properties (Zambonelli and Viroli, 2011), we consider chemistry following a series of work in the field of pervasive computing (Viroli and Zambonelli, 2010; Viroli *et al*, 2011; Zambonelli *et al*, 2011).

All such works can be framed under a common meta-model, describing a general mapping of computational abstractions into chemical abstractions. The basic idea is that system state depends on the kind and configuration of the 'molecules' floating in it, and system evolution (over space and time) depends on the different kinds of chemical reactions applicable. A more precise mapping can be done by assuming an agent-oriented viewpoint over computations, namely, considering an *agent* as an autonomous entity that (i) executes actions over the system according to the perception received from the environment, and (ii) encapsulates the strategy to choose among the set of available actions. Accordingly, agent state can be modelled as a set of 'molecules' and its autonomous internal behaviour is modelled via a set of chemical-like reactions. Agents perceive the environment by molecules coming from the environment and entering its boundaries. Given the agent state and perceptions only a sub-set of chemical reactions is enabled. The result of reaction execution is an action (or a set of actions), which may change the system state, for example, the configuration of molecules. Multiple agents, which constitute a *society*, can communicate by exchanging molecules with agents in the neighbourhood, namely, such molecules flow outside the boundary of an agent to enter another one. Finally, the *environment*, possibly continuous, is responsible of linking agents and defining their neighbourhood: a key role of the environment is hence in support diffusion of molecules, which should be carried on by proper chemical-like rules.

We observe that in order to be expressive enough to make the mapping fruitful, the chemical metaphor should be suitably enhanced:

- Molecules should be sort of data templates (ie, *annotations*), each one described not solely by a concentration value, but by a set of properties so that the system state may be complex and structured.
- The concept of reaction should be more elaborated than that used in chemistry: in classical chemical models, a reaction lists a number of reactant molecules that, combined, produce a set of product molecules. This kind of description shortly falls in practice (Ewald *et al*, 2007); it should be generalised to consider a reaction as a set of conditions about the system state, which trigger the execution of a set of actions. A condition is a function associating a boolean value to the current state of the system (or a subpart of it), and an action is a procedure that modifies it. Conditions are typically expressed as a list of annotations that must be available in a locality for the reaction to be executed, but might also include considerations about the shape of the neighbourhood (how and which agents are around, and whether they host certain annotations). Actions may be transformation, removal or production of annotations, agent movement, agent duplication and so on.
- The concept of neighbourhood should be as general and flexible as possible: from the physical concept of agents

inside a radius to the social concept of agents linked due to social relations of any kind—humans connected by a friendship relation.

- The speed of a classic reaction is described by a propensity function, which is fixed and depends only on the concentration of reactants (Gillespie, 1977). The propensity in computational systems should promote more flexibility: it is a function of the reaction rate, the conditions and the environment state.

Given the chemical inspiration, one would be tempted about predicting system behaviour reusing the existing body of chemical simulators (Gillespie, 1977; Gibson and Bruck, 2000) or of related stochastic computational models (Ewald *et al*, 2007), especially because of their known performance. However, they do not easily account for the above described enhancements: they manipulate only very simple data (eg, numeric concentrations associated to atomic molecules), have quite rigid atomic computation steps (addition/removal of molecules), rarely support multiple compartments, node mobility, and addition or removal of reactions and compartments at runtime (Alves *et al*, 2006).

It is with this consideration in mind that we decided to start from such chemical simulators for their performance, but properly enrich the chemical metaphor introducing concepts typical of situated computational systems.

## 3. ALCHEMIST

In this section we first introduce the computational model we defined for a simulator able to support the above meta-model, then show the details of the internal engine, and finally present some details about the overall framework, including its implementation and usage.

### *3.1. Computational model*

A pictorial representation of the underlying computational model is shown in Figure 1. In this simple vision of the world, an environment is a multi-dimensional space, continuous or discrete, which is able to contain nodes and is responsible of linking them following a rule. The environment may or may not allow nodes to move. Nodes are entities that can be programmed with a set of reactions possibly changing over time. They also contain molecules, each one equipped with a data structure generalising on the concept of 'concentration'.

The concept of reaction we use in the simulator is exactly the same as described in Section 2. This allows, for example, to model reactions that are faster if a node has many neighbours, or also reactions that resemble complex biological phenomena such as the diffusion of morphogenes during embryo development as described in Lecca *et al* (2010). It also allows to define which kind of time distribution to use to trigger reactions: this enables us to model and simulate systems based on Continuous Time Markov Chains (CTMCs), to add triggers, or also to rely just on classical discrete time 'ticks'.

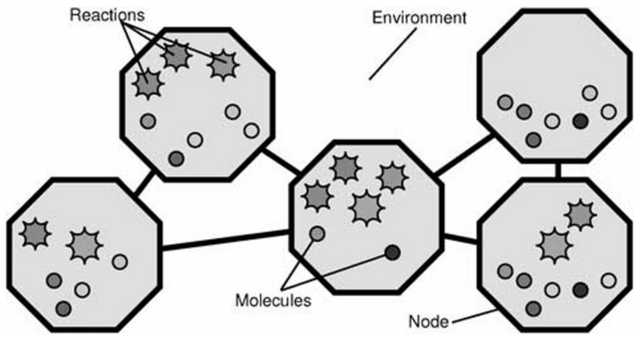

**Figure 1** ALCHEMIST computational model: it features a possibly continuous space embedding a linking rule and containing nodes. Each node is programmed with a set of reactions and contains a set of structured molecules.

### *3.2. Chemical engine*

The exact stochastic simulation algorithms for chemistry, originated in work of Gillespie (Gillespie, 1977), allows for finding exact possible evolutions of a chemical system, modelling each reaction event that takes place. When multiple compartments are considered, the set of reactions to be scheduled quickly explodes, and several methods have been studied to ensure high performance with an increasing number of reactions and molecular species. Today we have algorithms that run in logarithmic or even constant time with the number of reactions (Gibson and Bruck, 2000; Slepoy *et al*, 2008) and we hence aim at delivering similar performance in simulations of computational systems—provided the necessary level of flexibility in specifying system behaviour can be reached.

Our choice for ALCHEMIST was the Next Reaction Method (Gibson and Bruck, 2000). At each simulation step, it is able to select the next reaction to execute in constant time, and requires only logarithmic time to adjust its internal data structures after an update event. Such a choice was driven by two main factors. First, the algorithm in Gibson and Bruck (2000) does not choose the next reaction to execute by evaluating its speed as in the original Gillespie's version, but rather generates a putative time and smartly orders reactions to quickly choose the next to be executed—this scheduling model is much more flexible and is independent of the underlying stochastic model, which needs not be strictly that of Continuous-Time Markov Chains as in chemistry. Second, even though other algorithms with better performance exist (eg, Slepoy *et al*, 2008), they make such strong assumptions on system behaviour (eg, that the speed of reactions may differ of up to few orders of magnitude, or that the set of reactions will not change) whose impact on the computing scenarios of interest here are unclear and deserve further investigation.

According to the Next Reaction Method, then, ALCHEMIST follows the basic steps listed in Algorithm 1.

**Algorithm 1** Simulation flow in ALCHEMIST

```
1:  cur_time = 0
2:  cur_step = 0
3:  for each node n in environment do
4:     for each reaction nr in n do
5:        generate a new putative time for nr
6:        insert nr in DIPQ
7:        generate dependencies for nr
8:  while cur_time < max_time and cur_step < max_step do
9:     r = the next reaction to execute
10:    if r's conditions are verified then
11:       execute all the actions of r
12:       for each reaction rd which depends on r do
13:          update the putative execution time
14:    generate a new putative time for r
```

Likewise other simulation approaches for chemistry (Gillespie, 1977; Slepoy *et al*, 2008) Gibson's algorithm is not appropriate as-is to support our simulations: in particular, it does not provide facilities to add/remove/move nodes (which ultimately changes the set of reactions and their interdependencies dynamically), and to inject triggers and other non-exponential time distributed events. Hence, our work on the engine had the primary goal to extend the Next Reaction Method providing the possibility to add and remove reactions dynamically—an issue that, to the best of our knowledge, was never faced before. In order to add this support, it is mandatory to provide methods to add and remove reactions from two key data structures used in Next Reaction, namely, the indexed priority queue and the dependency graph, which are described in turn.

*(1) Dynamic Indexed Priority Queue*: The Indexed Priority Queue (IPQ) is a data structure proposed in Gibson and Bruck (2000). It is a binary tree of reactions, whose main property is that each node stores a reaction whose putative time of occurrence is lower than each of its sons. This means that the next reaction to execute is always in the root of the tree and can be accessed in constant time. A key property of the original IPQ is that the swap procedure used to update the data structure does not change the balance of the tree, ensuring optimal update times in every situation. This feature was easily achieved because no nodes were ever added neither removed from the structure. As a consequence, once the tree is balanced at creation time no event can occur to change its topology. This is no longer the case in ALCHEMIST, and we have to provide an extension to the IPQ machinery in order to properly handle tree balancing. (Figure 2)

Our idea is, for each node, to keep track of the number of descendant per branch, having in such way the possibility to keep the tree balanced when adding nodes. In Figure 3 we show how the same IPQ drawn in Gibson and Bruck (2000) would appear with our extension. Given this data structure, the procedures to add and remove a new node *n* are described, respectively, in Algorithms 2 and 3, in which the procedure `UPDATE_AUX (n)` is the same as described in Gibson and Bruck (2000).

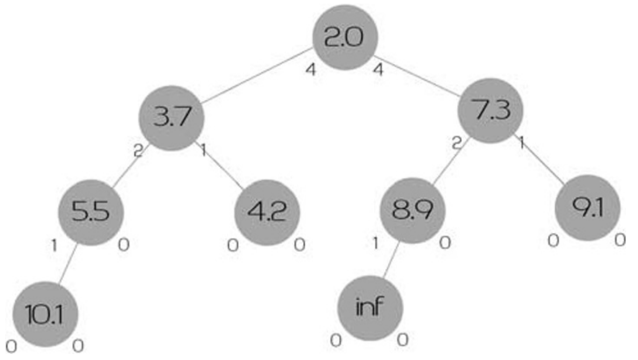

**Figure 3** Indexed priority queue extended with descendant count per branch.

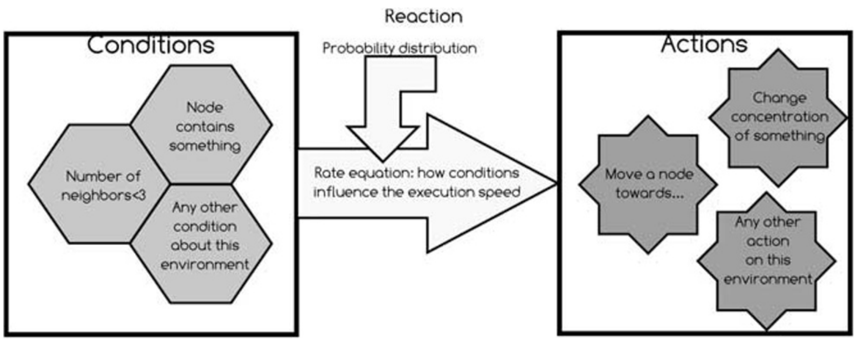
Reaction
Conditions
Probability distribution
Actions
Node contains something
Number of neighbors<3
Any other condition about this environment
Rate equation: how conditions influence the execution speed
Change concentration of something
Move a node towards...
Any other action on this environment

**Figure 2** ALCHEMIST model of reaction: a set of conditions on the environment that determines whether the reaction is executable, a rate equation describing how the speed of the reaction changes in response to environment modifications, a probability distribution for the event and a set of actions, which will be the neat effect of the execution of the reaction.

**Algorithm 2** Procedure to add a new node n

```
1:  if root does not exist then
2:     n is the new root
3:  else
4:     c ← root
5:     while c has two descendants do
6:        if c.right < c.left then
7:           dir ← right
8:        else
9:           dir ← left
10:       add 1 to count of dir descendants
11:       c ← c.dir
12:       if c has not the left child then
13:          n becomes left child of c
14:          set count of left nodes of c to 1
15:       else
16:          n becomes right child of c
17:          set count of right nodes of c to 1
18:    UPDATE_AUX (n)
```

**Algorithm 3** Procedure to remove a node n

```
1:  c ← root
2:  while c is not a leaf do
3:     if c.left < c.right then
4:        dir ← left
5:     else
6:        dir ← right
7:     subtract 1 to count of dir descendants
8:     c ← c.dir
9:  if c ≠ n then
10:    swap n and c
11:    remove n
12:    UPDATE_AUX (c)
13: else
14:    remove n
```

Using the two procedures described above, the topology of the whole tree is constrained to remain balanced despite the dynamic addition and removal of reactions.

*(2) Dynamic Dependency Graph*: The Dependency Graph (DG) is a directed graph in which nodes are triggered reactions, and arcs connect a reaction `r` to all those that depend on it, namely, those whose triggering time should be updated as *r* is executed: for instance, if `r` moves a molecule from a node `n` to another `m`, all those reactions that use the molecule in `m` are to be properly re-scheduled as soon as `r` is fired. Keeping the DG updated over time is one of the most critical task in the simulator, since it directly influences the speed at which we can affect simulation state each time a new action is executed.

Since we want to support natively and efficiently the interaction between nodes, which become dependencies among the reactions occurring in such nodes, we defined three contexts (also called scopes): `local`, `neighborhood` and `global`. Each reaction has an input context and an output context dynamically computed, which, respectively, represent where data influencing the rate calculus is located and where the modifications are made.

The first issue to address is to evaluate if a reaction `r1` may influence another reaction `r2`, considering their contexts. We introduced a boolean procedure `mayInfluence (r1, r2)`, which operates on two reactions and returns a true value if:

- `r1` and `r2` are on the same node OR
- `r1`'s output context is `global` OR
- `r2`'s input context is `global` OR
- `r1`'s output context is `neighborhood` and `r2`'s node is in `r1`'s node neighbourhood OR
- `r2`'s input context is `neighborhood` and `r1`'s node is in `r2`'s node neighbourhood OR
- `r1`'s output context and `r2`'s input context are both `neighborhood` and the neighbourhoods of their nodes have at least one common node.

Given this handy function, we can assert that a dependency exists between the execution of a reaction `r1` and another reaction `r2` if `mayInfluence (r1, r2)` is true and at least a molecule whose concentration is modified by `r1` is among those influencing `r2`.

Adding a new reaction implies to verify its dependencies against every reaction of the system. In case there is a dependency, it must be added to the DG. Removing a reaction r requires to delete all dependencies in which r is involved both as influencing and influenced. Moreover, in case of a change of system topology, a dependency check among reactions belonging to nodes with modified neighbourhood is needed. It can be performed by scanning them, calculating the dependencies with the reactions belonging to new neighbours and deleting those with nodes that are no longer in the neighbourhood.

## *3.3. Architecture*

The whole framework has been designed to be fully modular and extensible. The whole engine or parts of it can be re-implemented without touching anything in the model, and on the other hand the model can be extended and modified without affecting the engine.

It is important to note that there is no restriction about the kind of data structure representing the concentration, which can in fact be used to model structured information: by defining a new kind of structure for the concentration, it is possible to incarnate the simulator for different specific uses. For example, by assessing that the concentration is an integer number, representing the number of molecules currently present in a node, ALCHEMIST becomes a stochastic simulator for chemistry. A more complex example can be the definition of concentration as a tuple set, and the definition

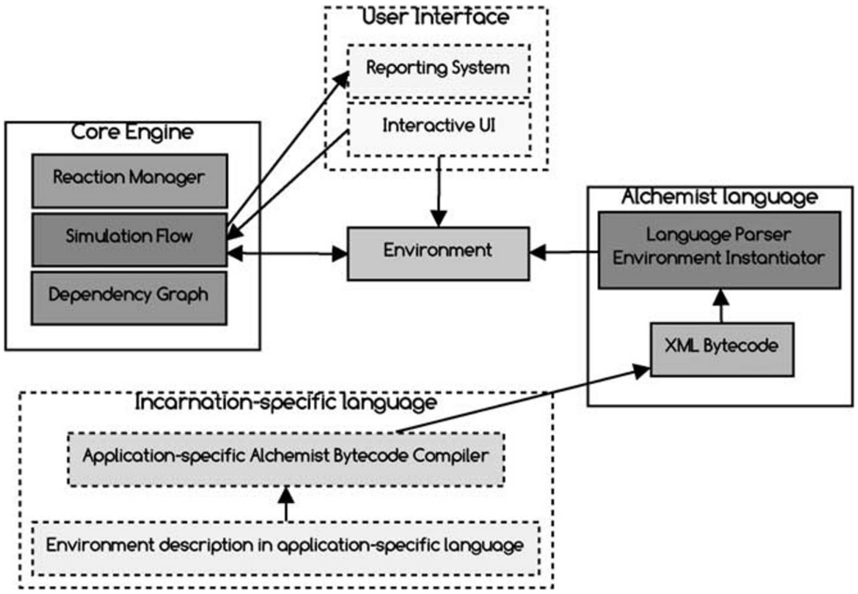

**Figure 4** ALCHEMIST architecture. Elements drawn with continuous lines indicates components common for every scenario and already developed, those with dotted lines are extension-specific components which have to be developed with a specific incarnation in mind.

of molecule as tuple template. If we adopt this vision, ALCHEMIST can be a simulator for a network of tuple spaces. Each time a new definition of concentration and molecule is made, a new 'incarnation' of ALCHEMIST is automatically defined. For each incarnation, a set of specific actions, conditions, reactions and nodes can be defined, and all the entities already defined for a more generic concentration type can be reused.

### 3.4. Implementation details

The framework was developed from scratch using Java. Being performance a critical issue for simulators, we compared some common languages in order to evaluate their performance level. Surprisingly, Java performance are at same level of compiled languages such as C/C++ (Bull *et al*, 2003; Oancea *et al*, 2011). The Java language was consequently chosen because of the excellent trade-off between performance, easy portability and maintainability of the code, and the high-level support for concurrent programming at the language level. The COLT Java library (Hoschek, 2004) provided us the mathematical functions we needed. In particular, it offers a fast and reliable random number generation algorithm, the so-called Mersenne Twister (Matsumoto and Nishimura, 1998). ALCHEMIST is still actively developed and currently consists of about 200 classes for about 20 000 lines of code. Though still in beta version, it is released with GPL license as open source.[1]

[1]http://alchemist.apice.unibo.it.

### 3.5. Writing a simulation

In order to write a simulation, the user must have, or implement herself, an incarnation of ALCHEMIST, as described in Section 3—Architecture.

As shown in Figure 4, the simulations are written in a specific XML language containing a complete description of environment and reactions. This code is interpreted in order to produce an instance of an environment: once it is created, no further interpretation is needed in order to run the simulation. This XML code is not meant to be directly exploited by users, but it represents a way of describing environments in a machine-friendly format and is a formalisation of the generic model of ALCHEMIST. The idea behind this choice is that ALCHEMIST is flexible enough to be used in various contexts, each one requiring a personalised language and a different instantiation of the model. It is up to the extensor to write a translation module from its personalised language to the ALCHEMIST XML. Of course, it is also possible to code the simulation behaviour directly with Java. Code examples for each language, related to the example presented in Section 4, are provided in Figure 5.

The usage of Java can be useful not only to run simulations, but also to exploit the engine to build standalone, easy-to-deploy simulations.[2]

## 4. Case study

In this section, we show the meta-model and framework at work in a pervasive computing scenario. We propose a

[2]An example, built on the code we used for the case study in Section 5, is available through the ALCHEMIST website.

**High-level, domain specific code:**

```
[<source,Id,Type,N>;] --> [<source,Id,Type,N>;<field,Id,Type,N,#T>;] rate "50"
```

**ALCHEMIST XML code:**

```
<reaction name="react0" type="LsaExpTimeReaction" p0="node" p1="50">
   <condition name="cond0" type="LsaStandardCondition" p0="source,Id,Type,N"></condition>
   <action name="act0" type="LsaStandardAction" p0="source,Id,Type,N"></action>
   <action name="act1" type="LsaStandardAction" p0="field,Id,Type,N,#T"></action>
</reaction>
```

**Java code:**

```
ILSAMolecule sourcePresent = new LsaExpMolecule("source,Type,Nmax,Type1");
ILSAMolecule genField = new LsaExpMolecule("field,Id,Type,N,#T");
IReaction<List<? extends ILSAMolecule>> r = new LsaExpTimeReaction(node, 50);
List<ICondition<? extends ILSAMolecule>> c = new ArrayList<ICondition<? extends ILSAMolecule>>(1);
List<IAction<? extends ILSAMolecule>> a = new ArrayList<IAction<? extends ILSAMolecule>>(1);
ILsaCondition c1 = new LsaStandardCondition(sourcePresent,node);
ILsaAction a1 = new LsaStandardAction(sourcePresent,node);
ILsaAction a2 = new LsaStandardAction(genField,node);
c.add(c1); a.add(a1); a.add(a2);
r.setConditions(c); r.setActions(a);
```

**Figure 5** Code translation of the first eco-law. The domain specific language is obviously easier to understand and dramatically more compact with respect to the other forms.

crowd steering case study in which a middleware has the goal of leading people in the desired location within a complex environment in short time (Viroli and Zambonelli, 2010; Zambonelli and Viroli, 2011) avoiding obstacles such as crowded regions and without global supervisioning.

Consider a museum with a set of rooms, whose floor is covered with a network of computational devices (infrastructure nodes). These devices can exchange information with each other based on proximity, sense the presence of visitors and hold information about expositions currently active in the museum. Each room has four exits and they are connected via external corridors. Visitors wandering the museum are equipped with a hand-held device that holds the visitor's preferences. By interaction with infrastructure nodes, a visitor can be guided towards rooms with a target matching their interest, thanks to signs dynamically appearing on his smartphone or on public displays. This is done using techniques suggested in the field of spatial computing (Viroli *et al*, 2011)—namely, computational gradients injected in a source and diffusing around such that each node holds the minimum distance value from the source.

This kind of problems, strictly related with spatiality, can be tackled in different ways (Murphy *et al*, 2006; Roy *et al*, 2008; Autili *et al*, 2009; Fok *et al*, 2009; Mamei and Zambonelli, 2009). We here endorse the general idea of Viroli *et al* (2011), and assume each computational device holds a space of annotations (structured tuples) inserted/removed by agents running in that node. Computation is made in a self-organising way by basic chemical-like laws evolving the population of annotations, enacting mechanisms of coordination, communication and interaction, as in Viroli and Zambonelli (2010). This approach is also in line with Viroli *et al* (2011) in which the annotations are referred to as Live Semantic Annotations (LSA), and chemical-like reactions are called eco-laws to remind of their role being similar to that of laws of an ecosystem.

## *4.1. Modelling the pervasive system*

The environment is a continuous bidimensional space with walls. Smartphones (or public displays) are agents dynamically linked with the nearest infrastructure node—the neighbours are the sensors inside a certain radius *r*, parameter of the model—from which they can retrieve data in order to suggest visitors where to go. Visitors are agents who follow the advices of their hand-held device (or public displays). It is defined a minimum possible distance between them, so as to model the physical limit and the fact that two visitors can't be in the same place at the same time. Visitors can move of fixed size steps inside the environment. If an obstacle is on their path, their movement is shortened to the allowed position nearest to the desired place.

All the information exchanged is in form of annotations, simply modelled as tuples $\langle v_1, \ldots, v_n \rangle$ (ordered sequence) of typed values, which could be for example numbers, strings, structured types or function names. There are three forms of annotations used in this scenario:

$$\langle \text{source}, id, type, N_{\max} \rangle$$
$$\langle \text{field}, id, type, value, tstamp \rangle$$
$$\langle \text{info}, id, crowd, M, t' \rangle$$

A `source` annotation is used as a source with the goal of generating a field: *id* labels the source so as to distinguish

$$
\begin{array}{rcl}
\langle \texttt{source}, id, type, N_{max} \rangle & \xmapsto{r_{init}} & \langle \texttt{source}, id, type, N_{max} \rangle, \langle \texttt{field}, id, type, N_{max}, \#T \rangle \\
\langle \texttt{field}, id, type, N, t \rangle & \xmapsto{r_{diff}} & \langle \texttt{field}, id, type, N, t \rangle, +\langle \texttt{pre\_field}, id, type, N - \#D, t \rangle \\
\langle \texttt{pre\_field}, id, type, N, t \rangle, \langle \texttt{info}, id, crowd, M, t' \rangle & \mapsto & \langle \texttt{field}, id, type, N - k * C, t \rangle, \langle \texttt{info}, id, crowd, M, t' \rangle \\
\langle \texttt{field}, id, type, N, t \rangle, \langle \texttt{field}, id, type, M, t' + t \rangle & \mapsto & \langle \texttt{field}, id, type, M, t' + t \rangle \\
\langle \texttt{field}, id, type, N, t \rangle, \langle \texttt{field}, id, type, M, t \rangle & \mapsto & \langle \texttt{field}, id, type, max(M, N), t \rangle \\
\langle \texttt{field}, id, type, N, t \rangle, \langle \texttt{field}, id', type, N + M, t' \rangle & \mapsto & \langle \texttt{field}, id', type, N + M, t' \rangle
\end{array}
$$

**Figure 6** Laws describing the museum application.

sources of the same type; *type* indicates the type of fields in order to distinguish different expositions; $N_{\max}$ is the field's maximum value. A `field` annotation is used for individual values in a gradient: *value* indicates the individual value; the *tstamp* reflects the time of creation of the annotation; the other parameters are like in the source annotation. An `info` annotation is supposed to be created and kept up to date by each sensor. $M$ represents the number of smartphones the sensor is perceiving as neighbours.

The rules are expressed in form of chemical-resembling laws, working over patterns of annotations. One such pattern $P$ is basically an annotation that may have some variable in place of one or more arguments of a tuple, and an annotation $L$ is matched to the pattern $P$ if there exists a substitution of variables which applied to $P$ gives $L$. A law is hence of the kind $P_1, \ldots, P_n \overset{r}{\mapsto} P'_1, \ldots, P'_m$ , where: (i) the left-hand side (reagents) specifies patterns that should match annotations $L_1, \ldots, L_n$ to be extracted from the local annotation space; (ii) the right-hand side (products) specifies patterns of annotations, which are accordingly to be inserted back in the space (after applying substitutions found when extracting reagents, as in standard logic-based rule approaches); and (iii) rate $r$ is a numerical positive value indicating the average frequency at which the law is to be fired—namely, we model execution of the law as a CTMC transition with Markovian rate (average frequency) $r$. If no rate is given the reaction is meant to be executed 'as soon as possible', which means that the rate associated with the reaction tends to be infinite. To allow interaction between different nodes (hence, annotation spaces), we introduce the concept of *remote pattern*, written $+P$, which is a pattern that will be matched with an annotation occurring in a neighbouring space.

Figure 6 shows the laws for our case study. As sources annotations are injected in nodes, gradients are built by the first three rules. The first one, given a source, initiates the field with its possible maximum value. The second one, when a node contains a field annotation, spreads a copy of it to a neighbouring node picked up randomly with a new value computed considering the crowding around the sensor. The parameter $k$ allows to tune how much crowding should influence the field, while $\#D$ is the measure of the distance between the two involved sensors. As a consequence of these laws, each node will carry a field annotation indicating the topological distance from the source. The closest is the field value to $N_{\max}$, the nearest is the field source. When the spread values reach the minimum value 0, the gradient has to become a plateau.

To address the dynamism of the scenario where people move, targets are possibly shifted, and crowds form and dissolve, we introduced the following mechanism. We expect that if a gradient source moves the diffused value has to change according to the new position. This is the purpose of the *tstamp* parameter, which is used in the fourth law, continuously updating old values by more recent ones (*youngest* law). In this way we ensure that the system is able to adapt to changes of the source states. Finally, the spreading law above will produce duplicate values in locations, due to multiple sources of the same type (indicated by different ids), multiple paths to a source, or even diffusion of multiple annotations over time. For this reason we introduced the last two laws. They retain only the maximum value, that is the minimum distance, the former when there are two identical annotations with only a different value, the latter when the id is different (*shortest* laws).

The proposed solution is intrinsically able to dynamically adapt to unexpected events (like node failures, network isolation, crowd formation, and so on) while maintaining its functionality.

## 4.2. Simulator configuration

In order to simulate such environment with ALCHEMIST, a proper incarnation must be defined. In this case, we defined a molecule as an annotation template, and a concentration as a set of annotations. This means that for each annotation template in a node there will be a (possibly empty) set of matching annotations.

People are modelled as nodes programmed with a single reaction with no conditions and having, as action, the ability to follow the highest field value among neighbouring sensors.

The behaviour of each node is programmed according to the laws explained in Figure 6. Each law is modelled as a reaction whose conditions are the presence of annotations matching the templates on the left side of the law, and whose actions are the retrieval of the matched annotations and the insertion of new ones on the right side.

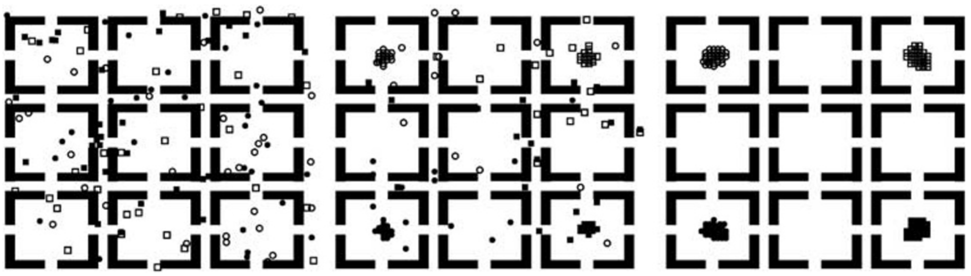

**Figure 7** A simulation run of the reference exposition: three snapshots of the ALCHEMIST graphic reporting module with this simulation.

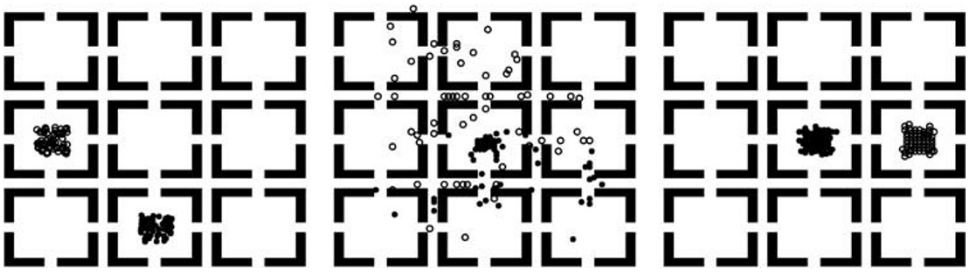

**Figure 8** A run showing the effect of crowding: dark visitors occupy a central room, making other visitors moving left to right by a longer, less crowded path.

The sources of the gradients are injected by sensors when a target is perceived. For the crowding annotation instead, we may assume that sensors are calibrated so as to locally inject and periodically update it by setting the current level of crowding, that is the number of persons.

The reaction rates are identified by hand performing different simulations with different parameters. The results reported are obtained with $r_{init} = 1$ and $r_{diff} = 50$. The other laws show no rate because it is assumed to be infinite.

### *4.3. Simulation results*

We here present simulations conducted over an exposition, where nine rooms are connected via corridors. People can express different preferences represented by their shape.

Three snapshots of a first simulation run are reported in Figure 7. We here consider four different targets that are located in the four rooms near environment angles. People are initially spread randomly in the museum, as shown in the first snapshot, and they eventually reach the room in which the desired target is hosted, as shown in the last snapshot.

Figure 8 shows a simulation experimenting with the effect of crowding in the movement of people. Two groups of people—denoted with empty and full circles—with common interests are initially located in two different rooms, as shown in the first snapshot. The target for the dark visitors is located in the central room of the second row, while the others' is in the right room of the second row. In the simulation, dark visitors reach their target soon because it is nearer, thus forming a crowded area intersecting the shortest path towards the target for the other visitors. Due to this jam the latter visitors choose a different path that is longer but less crowded.

Both tests show qualitative effectiveness of the proposed laws, and suggest that our simulation approach can be used for additional experiments focussing on tuning system parameters (factor $k$) or alternative strategies (eg, diffusing crowd information) to optimise paths to destinations. For instance, in the context of the second case, Figure 9 shows how factor $k$ can influence the time for (sub)groups of (light) people to reach the destination, by which we can see that even small values of $k$ lead to a significant improvement—which slowly decreases as $k$ grows.

## 5. Comparison with existing works

### *5.1. ALCHEMIST as a discrete event simulator (DES)*

ALCHEMIST is a DES, since it combines a continuous time base with the description of system dynamics by distinguished state changes (Zeigler, 1976). Since it allows for in-simulation modifications of the environment, it can be considered to belong to the fourth generation of DESs according to Babulak and Wang (2008). The work in Pollacia (1989) surveys the classical DES approaches: according to it, ALCHEMIST belongs to the category of simulators featuring 'internal clock', 'next-event time advance' and adhering to the 'Event-scheduling World View', namely, the

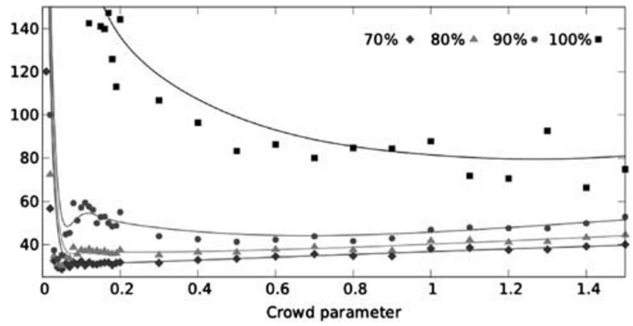

**Figure 9** Time units of convergence time with different values of crowd parameter and different percentages of people.

simulator handles events and is concerned with their effect on system states.

Apart from the meta-model adopted, the main innovative aspect of ALCHEMIST with respect to the general DES approach is its ability of optimising the 'compile current event list' stage (Pollacia, 1989), which ALCHEMIST quickly executes incrementally by means of the management of dependencies that the adoption of a chemical-like model enables.

As far as the simulator model is concerned, instead, the class of DES more related to our approach are those commonly used to simulate biological-like systems. A recent overview of them is available in Ewald *et al* (2007), which takes into account: DEVS (Zeigler, 1984), Petri Nets (Murata, 1989), State Charts (Harel 1987) and stochastic $\pi$-calculus (Priami, 1995). However, such an overview emphasises that all such models require a considerable effort to map biological components into abstractions of the chosen formalism: this is because none of them was specifically developed with biology or bio-inspiration in mind. As described in Section 2, our model is meant to overcome such limitations, since all the enhancements to the basic chemical model we support can be seen as a generalisation of the abstractions of the works presented in Ewald *et al* (2007), and add to them the possibly of customising with much greater flexibility a simulation to the scenarios of bio-inspired computational systems.

We should finally note that the DES approach typically contrasts the use of mathematical techniques (eg modelling the system by differential equations). However, the possible different choice of translating a system model to ordinary or partial different equations—which can be solved numerically in a time considerably shorter than a set of stochastic simulations—is shown to be impractical in our case. This path, explored for example by Mallavarapu *et al* (2009), can provide good results for dynamics that progress at more or less the same speed, and in which abundance of species (data items or agents in our case) is so high that it can be relaxed to real-valued variables (Ewald *et al*, 2007). This is not the case for many scenarios even in system biology (see, for example, Cowan, 2000; Uhrmacher and Priami, 2005), not to mention scenarios like pervasive computing where reaction rates do not change continuously, and where the effect on an even small number of agents can be key to a given system evolution.

## 5.2. *ALCHEMIST as a multi-agent-based simulation (MABS)*

Even though chemical-inspired, the agent-based meta-model described in Section 2 is a natural one to reason about how a simulation can be structured in ALCHEMIST. As such, relationships with MABS approaches is evident—most of which are easily categorised as DESs as well.

According to Bandini *et al* (2009a), agent-based platforms for simulations can be split in three categories: general purpose frameworks with specific languages such as NetLogo, general purpose frameworks based on general purpose programming languages such as (North *et al*, 2007) and frameworks that provide an higher level linguistic support, often targeted to a very specific application (eg Weyns *et al*, 2006). Each approach has clearly its own advantages and weaknesses. Usually, the more general purpose is the language, the wider is the set of possible scenarios, and the wider is also the gap between the language and the simulated model. This means that a higher level language allows the user to create and tune its simulations in an easier way, on the other hand it often restricts the generality of the tool.

ALCHEMIST is meant to provide a way to drive the simulator with a high level language of chemical reactions (third approach above), still maintaining the possibility to extend or re-implement certain abstractions using the general purpose Java language (second approach above). Namely, the idea is to make as easy as possible to embed Java-written functionalities into ALCHEMIST concepts, as exemplified in previous section. This also supports the possibility of writing new domain-specific languages every time a new extension is made: this allows both extensibility and quick deployment of simulations—ALCHEMIST current release embeds one such language, tailored to the framework in Zambonelli *et al* (2011).

Accordingly, as far as simulating systems equipped with an ABM is concerned, there is a set of applications that are better tackled by ALCHEMIST. Namely, ALCHEMIST is suitable for all those simulation scenarios in which agents have relatively simple behaviour, and the notion of agent-based environment plays instead a fundamental role in organising and regulating the agents' activities (Bandini and Vizzari, 2007) by both enabling local interactions among the proactive entities (Helleboogh *et al*, 2007) and enforcing coordination rules (Molesini *et al*, 2009), allowing the modeller to shift her focus from the local behaviour of the single agent to a more objective vision of the whole MAS (Schumacher *et al*, 2007). The idea of dealing with a strong notion of environment in multi-agent systems has been deeply developed in a series of work: other than its importance in the simulation context (Helleboogh *et al*, 2007), at the infrastructure level (Viroli *et al*, 2007) and at the methodological level (Molesini *et al*, 2009), there have been proposals of meta-models such as A&A (Omicini *et al*, 2008) and infrastructures such as TuCSoN (Omicini and Zambonelli, 1999) and CartAgO (Ricci *et al*, 2011). A common viewpoint of all these works is that the behaviour of those passive and reactive components structuring the environment (eg *artifacts* in A&A) is well defined in terms of rules expressed as declarative conditions-imply-actions fashion. Accordingly, ALCHEMIST is particularly useful either in computing systems following the chemical inspiration (Viroli and Zambonelli, 2010; Viroli *et al*, 2011; Zambonelli *et al*, 2011) or for agent-oriented systems where the role of the environment is key, up to be a very dynamic part of the whole system—where network nodes (or, in biological terms, compartments) can move, new nodes can be spawn or be removed from the system, and links can appear or break at runtime as happen eg in pervasive computing scenarios (Zambonelli *et al*, 2011).

Inevitably, as an attempt to build a hybrid between agent-based simulators and (bio)chemical simulators, some trade-offs had to be accepted, which ultimately makes ALCHEMIST less suited for certain classes of applications. In particular, a limitation is that ALCHEMIST's agent actions have to be mapped onto the concept of reaction. On the one hand, this makes it rather straightforward to create reactive memoryless agents (Bandini *et al*, 2009a), whose goal is just to perform rather easy computations resulting in the creation, deletion or modification of information items in the environment. In fact, since it is allowed to programme different nodes with different reactions, it is easy to code reactive and context-dependent agents. On the other hand, there is neither out-of-the-box facility nor any high level abstraction useful to define intelligent agents (Bandini *et al*, 2009a). The simulator is able to run them provided the user manually writes their whole behaviour as a single Java-written reaction—and properly specifies dependencies with other reactions. Although possible in principle, performing this task too frequently will likely break the ALCHEMIST abstraction, making the programmer lose the nice declarative approach that chemistry endorses, and possibly hamper the optimisation techniques that motivated ALCHEMIST.

Another limitation that worth being mentioned concerns the availability of tools to visualise and analyse a simulation, which are currently much more sophisticated in frameworks such as MASON (Luke *et al*, 2005), Repast (North *et al*, 2007), NetLogo (Sklar, 2007) and Swarm (Minar et al, 1996). At the current development stage ALCHEMIST does not provide a complete GUI or analysis tools, even if some useful tools are already in their place. This is not to be seen as a limitation of the general ALCHEMIST approach, but rather a consequence of its current development/maintenance model.

### *5.3. Performance*

It is possible to evaluate and compare the performances of ALCHEMIST with respect to some known MABS. We exemplify it considering Repast, which we used to developed an alternative simulation for the case study presented in Section 4. There are some important facts that deserve discussion here. First, since there is no built-in support for stochastic simulation in Repast, we choose to collapse the last five laws of Figure 6 into a single code path, and the same was made for ALCHEMIST by defining a new action. In that way, we were able to avoid for this very specific case the need of a full-fledged DG, because there will always be exactly one `source` and one `field` per sensor, and no reactions need to be enabled or disabled. This crippled most of the effectiveness of the ALCHEMIST DG, which is indeed a source of optimisation not natively existing in Repast—developing a DG manager for stochastic simulation in Repast is out of scope here, though it would be an interesting subject for future work. On the other hand, it would have been unfair to compare our optimised version against just a plain Gillespie's algorithm built upon Repast. The second important point is that we chose to encode all the data in both Repast and ALCHEMIST as an array of double values instead of real tuples. This was done because the pervasive computing incarnation of ALCHEMIST, which allows us to write the laws as in Figure 6, requires a matching system that is not easily portable to Repast. The choice of encoding data that way made things faster (no matching required), but also less general.

We chose the configuration of Figure 8 and we ran multiple simulations varying the number of agents per group, in order to evaluate how the two systems scale with the problem size. We measured the running time required to our testbed to run the simulation from the time zero to the time 100. Obviously, no graphical interface were attached to the simulators while running the batch, in order to evaluate only the raw performance of the engine. The system we used was an Intel Core i5-2500 equipped with 8 GB RAM, with Sabayon Linux 7 as operating system and Sun Java HotSpot™ 64–Bit Server version 1.6.0_26–b03 as Java Virtual Machine.

Simulation results are reported in Figure 10, and show the simulator built upon the ALCHEMIST framework to be at least twice faster and to scale better than the one built on Repast. Being the DG optimisation cut-off as explained above, the reasons for such a difference can lay on the internal scheduler of the engine or in the optimisations in the model. For the first point, we used the default scheduler of Repast Symphony, which is a binary tree implemented through a plain Java array, while our implementation relies on the algorithm and data structures already presented in Section 3, so there is not a substantial efficiency difference.

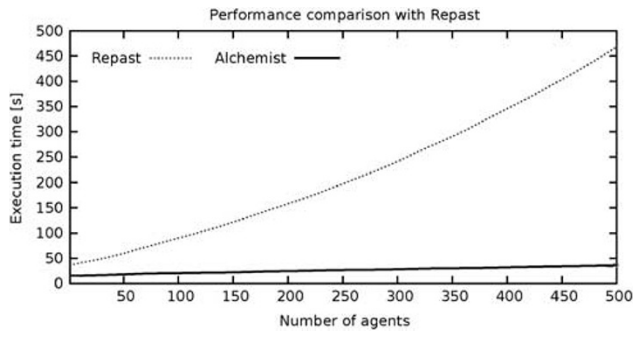

**Figure 10** Chart showing the performance scaling of ALCHEMIST.

For the latter point, a big difference in terms of performances is due to the heavy optimisations of the neighbourhoods of the default ALCHEMIST continuous environment. Since the concept of neighbourhood was part of the computational model, it was possible to adopt caching strategies in order to ensure a fast access to the neighbourhood and a quick execution of the operations on it. This is probably the component that gives the highest impact in this case, since most interactions occur among an agent and its neighbourhood.

## 6. Conclusion

Complex systems are often easy to describe in terms of communicating entities inside an environment. A possible abstraction for their behaviour is to decompose it in terms of atomic events resembling chemical reactions, which can happen if some requirements are satisfied, generating some effects on the environment.

In this paper, we presented the ALCHEMIST simulation framework, capturing this abstraction and offering a fast and optimised engine that can be exploited to build specific incarnations of the generic model. We described the engine showing how we extended the existing work on stochastic simulation algorithms in order to deal with an highly dynamic environment, in which reactions can be dynamically added and removed from the system. This extension involves both the structure responsible of ordering the reactions and the one responsible of detecting dependencies among them. We then presented a case study in pervasive ecosystems, showing how the simulator is able to capture the expressiveness of a complex crowd steering scenario. We used it also to tune a parameter of our model. Finally, we compared our work with existing simulators, describing both strengths and weaknesses of this approach. Summarising, we produced a simulation framework that can run faster than standard MABS (eg, we compared with Repast), but forces the user to make her model adhere to the ALCHEMIST abstract one.

As emphasised throughout the paper, the approach proposed is meant to be used for modelling purposes in different domains. For instance, a challenging and promising field of application is computational biology. A first work towards this direction is presented in Montagna *et al* (2012). There, a developmental biology scenario is modelled and a significant set of mechanisms—nuclear division and molecular processes such as morphogen diffusion and gene regulatory networks—involved in *Drosophila M*. embryo development, are captured. Simulations in ALCHEMIST are run, reproducing the spatial pattern formation of gene expression observable in data acquired from the real system development.

In future work we will support other domain-specific languages to code simulations, and will focus on the interface, providing ways to observe and interact with simulations in a uniform way.

*Acknowledgements*—This work has been supported by the EU-FP7-FET Proactive project SAPERE Self-aware Pervasive Service Ecosystems, under contract no.256873

## References

Alves R, Antunes F and Salvador A (2006). Tools for kinetic modeling of biochemical networks. *Nature Biotechnology* **24**(6) pp 667–672.

Autili M, Benedetto P and Inverardi P (2009). Context-aware adaptive services: The plastic approach. In: *FASE'09 Proceedings*. Springer-Verlag, Berlin, Heidelberg, pp 124–139.

Babaoglu Ö *et al* (2006). Design patterns from biology for distributed computing. *TAAS* **1**(1): 26–66.

Babulak E and Wang M (2008). Discrete event simulation: State of the art. *iJOE* **4**(2): 60–63.

Banâtre J-P, Fradet P and Métayer DL (2001). Gamma and the chemical reaction model: Fifteen years after. In: *Proceedings of the Workshop on Multiset Processing: Multiset Processing, Mathematical, Computer Science, and Molecular Computing Points of View*, ser. WMP'00. Springer-Verlag, London, UK, pp 17–44.

Banâtre J-P and Métayer DL (1993). Programming by multiset transformation. *Communications of the ACM* **36**(1): 98–111.

Bandini S, Manzoni S and Vizzari G (2009a). Agent based modeling and simulation: An informatics perspective. *Journal of Artificial Societies and Social Simulation* **12**(4): 4.

Bandini S, Manzoni S and Vizzari G (2009b). Crowd behavior modeling: From cellular automata to multi-agent systems. In: Uhrmacher AM and Weyns D (eds). *Multi-agent Systems: Simulation and Applications* ser. Computational Analysis, Synthesis, and Design of Dynamic Systems, CRC Press, Taylor Francis Group, Boca Raton, FL 33487-2742, pp 389–418.

Bandini S and Vizzari G (2007). Regulation function of the environment in agent-based simulation. In: *Proceedings of the 3rd International Conference on Environments for Multi-agent Systems III*, ser. E4MAS'06, Springer-Verlag, Berlin, Heidelberg, 2007, pp 157–169.

Bergstra JA and Bethke I (2002). Molecular dynamics. *Journal of Logic and Algebraic Programming* **51**(2): 193–214.

Berry G and Boudol G (1992). The chemical abstract machine. *Theoretical Computer Science* **96**(1): 217–248.

Brijder R, Ehrenfeucht A, Main MG and Rozenberg G (2011). A tour of reaction systems. *International Journal of Foundations of Computer Science* **22**(7): 1499–1517.

Bull JM *et al* (2003). Benchmarking Java against C and Fortran for scientific applications. *Concurrency and Computation: Practice and Experience* **15**(3–5): 417–430.

Cardelli L and Gordon AD (2000). Mobile ambients. *Theoretical Computer Science* **240**(1): 177–213.

Cowan R (2000). Stochastic models for DNA replication. In: Shanbhag DN and Rao CR (eds). *Stochastic Processes: Modelling and Simulation*, Vol. 21. Handbook of Statistics. Elsevier, pp 137–166.

Credi A *et al* (2007). Modelization and simulation of nano devices in {*nano k*} calculus. In: *Computational Methods in Systems Biology, International Conference, CMSB 2007,* Edinburgh, Scotland, 20–21 September, Proceedings, ser. Lecture Notes in Computer Science, Vol. 4695. Springer, pp 168–183.

Ewald R, Maus C, Rolfs A and Uhrmacher AM (2007). Discrete event modeling and simulation in systems biology. *Journal of Simulation* **1**(2): 81–96.

Fernandez-Marquez JL *et al* (2012). Description and composition of bio-inspired design patterns: A complete overview. *Natural Computing*, doi: 10.1007/s11047-012-9324-y, online First, pp 1–25.

Fok C-L, Roman G-C and Lu C (2009). Enhanced coordination in sensor networks through flexible service provisioning. In: Field, J and Vasconcelos, VT (eds). *Proceedings of COORDINATION 2009* ser. LNCS, Vol. 5521, Springer-Verlag, Lisbon, Portugal.

Gardelli L, Viroli M and Omicini A (2007). Design patterns for self-organising systems. In: Verbrugge, R and Varga, LZ (eds). *Proceedings of the Multi-agent Systems and Applications V*, ser, *LNAI, Burkhard*, H-D. Springer, September, Vol. 4696, pp 123–132, *5th International Central and Eastern European Conference on Multi-agent Systems (CEEMAS'07)*, Leipzig, Germany, 25–27 September.

Gibson MA and Bruck J (2000). Efficient exact stochastic simulation of chemical systems with many species and many channels. *Journal of Physical Chemistry A* **104**(9): 1876–1889.

Gillespie DT (1977). Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry* **81**(25): 2340–2361.

Harel D (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming* **8**(3): 231–274.

Helleboogh A, Vizzari G, Uhrmacher A and Michel F (2007). Modeling dynamic environments in multi-agent simulation. *Autonomous Agents and Multi-agent Systems* **14**(1): 87–116.

Hoschek W (2004). *The Colt Distribution: Open Source Libraries for High Performance Scientific and Technical Computing in Java*. CERN, Geneva: http://acs.lbl.gov/software/colt/.

Lecca P, Ihekwaba AEC, Dematté L and Priami C (2010). Stochastic simulation of the spatio-temporal dynamics of reaction-diffusion systems: The case for the bicoid gradient. *Journal of Integrative Bioinformatics* **7**(1): 1–32.

Luke S *et al* (2005). Mason: A multiagent simulation environment. *Simulation* **81**(7): 517–527.

Macal CM and North MJ (2010). Tutorial on agent-based modelling and simulation. *Journal of Simulation* **4**(3): 151–162.

Mallavarapu A, Thomson M, Ullian B and Gunawardena J (2009). Programming with models: Modularity and abstraction provide powerful capabilities for systems biology. *Journal of the Royal Society, Interface/the Royal Society* **6**(32): 257–270.

Mamei M and Zambonelli F (2009). Programming pervasive and mobile computing applications: The TOTA approach. *ACM Transactions on Software Engineering and Methodology* **18**(4): 1–56.

Matsumoto M and Nishimura T (1998). Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modelling and Computer Simulation* **8**(1): 3–30.

Minar N, Burkhart R, Langton C and Askenazi M (1996). The {S}warm simulation system: A toolkit for building multi-agent simulations. Working Paper 96-06-042, Santa Fe, CA.

Molesini A, Casadei M, Omicini A and Viroli M (2011). Simulation in agent-oriented software engineering: The SODA case study. Science of Computer Programming, August 2011, Special Issue on Agent-oriented Design methods and Programming Techniques for Distributed Computing in Dynamic and Complex Environments. [Online]: http://www.sciencedirect.com/science/article/pii/S0167642311001778.

Molesini A, Omicini A and Viroli M (2009). Environment in agent-oriented software engineering methodologies. *Multiagent and Grid Systems* **5**(1): 37–57, Special Issue 'Engineering Environments in Multi-agent Systems'.

Montagna S, Pianini D and Viroli M (2012). A model for *Drosophila melanogaster* development from a single cell to stripe pattern formation. In: *27th Annual ACM Symposium on Applied Computing (SAC 2012)*, Riva del Garda (Trento), Italy, 26–30 March, to appear.

Murata T (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, Vol. **77**, pp 541–580.

Murphy AL, Picco GP and Roman G-C (2006). Lime: A model and middleware supporting mobility of hosts and agents. *ACM Transactions on Software Engineering and Methodology* **15**(3): 279–328.

North MJ, Howe TR, Collier NT and Vos JR (2007). A declarative model assembly infrastructure for verification and validation. In: Takahashi S, Sallach D and Rouchier J (eds). *Advancing Social Simulation: The First World Congress*. Springer: Japan, pp 129–140.

Oancea B, Rosca IG, Andrei T and Iacob AI (2011). Evaluating Java performance for linear algebra numerical computations. *Procedia Computer Science* **3**: 474–478.

Omicini A, Ricci A and Viroli M (2008). Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-agent Systems* **17**(3): 432–456 December, Special Issue on Foundations, Advanced Topics and Industrial Perspectives of Multi-agent Systems.

Omicini A and Zambonelli F (1999). Coordination for internet application development. *Autonomous Agents and Multi-agent Systems* **2**(3): 251–269.

Paun G (2002). *Membrane Computing: An Introduction*. Springer-Verlag, New York: Secaucus, NJ.

Pollacia LF (1989). A survey of discrete event simulation and state-of-the-art discrete event languages. *SIGSIM Simulation Digest* **20**(3): 8–25.

Priami C (1995). Stochastic pi-calculus. *The Computer Journal* **38**(7): 578–589.

Ricci A, Piunti M and Viroli M (2011). Environment programming in multiagent systems—An artifact-based perspective. *Autonomous Agents and Multi-agent Systems* **23**(2): 158–192.

Roy PV *et al* (2008). Self-management for large-scale distributed systems: An overview of the selfman project. In: Bonsangue MM, Graf S and de Roever W-P (eds). *Formal Methods for Components and Objects*, LNCS No. 5382. Springer Verlag, pp 153–178.

Schumacher M, Grangier L and Jurca R (2007). Governing environments for agent-based traffic simulations. In: *Proceedings of the 5th international Central and Eastern European* *conference on Multi-agent Systems and Applications V*, ser. CEEMAS'07. Springer-Verlag, Berlin, Heidelberg pp 163–172.

Sklar E (2007). Netlogo, a multi-agent simulation environment. *Artificial Life* **13**(3): 303–311.

Slepoy A, Thompson AP and Plimpton SJ (2008). A constant-time kinetic Monte Carlo algorithm for simulation of large biochemical reaction networks. *The Journal of Chemical Physics* **128**(20): 205101.

Uhrmacher AM and Priami C (2005). Discrete event systems specification in systems biology—A discussion of stochastic pi calculus and devs. In: *Proceedings of the 37th Conference on Winter Simulation*, ser. WSC'05. Winter Simulation Conference, pp 317–326.

Viroli M and Casadei M (2009). Biochemical tuple spaces for selforganising coordination. In: Field LJ and Vasconcelos VT (eds). *Proceedings of the Coordination Languages and Models*, ser., Lisbon, Portugal: Springer, June, Vol. 5521, pp 143–162, *Proceedings of the 11th International Conference (COORDINATION 2009)*.

Viroli M, Casadei M, Montagna S and Zambonelli F (2011). Spatial coordination of pervasive services through chemical-inspired tuple spaces. *ACM Transactions on Autonomous and Adaptive Systems* **6**(2): 14:1–14:24.

Viroli M *et al* (2007). Infrastructures for the environment of multiagent systems. *Autonomous Agents and Multi-agent Systems* **14**(1): 49–60.

Viroli M *et al* (2011). A coordination approach to adaptive pervasive service ecosystems. In: *1st Awareness Workshop 'Challenges in Achieving Self-awareness in Autonomous Systems' (AWARE 2011)*, SASO 2011, Ann Arbor, MI, 7 October.

Viroli M and Zambonelli F (2010). A biochemical approach to adaptive service ecosystems. *Information Sciences* **180**(10): 1876–1892.

Weyns D, Boucké N and Holvoet T (2006). Gradient field-based task assignment in an AGV transportation system. In: *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, ser. AAMAS'06, ACM, New York, NY, pp 842–849.

Zambonelli F *et al* (2011). Self-aware pervasive service ecosystems. *Procedia Computer Science* **7**: 197–199, *Proceedings of the 2nd European Future Technologies Conference and Exhibition 2011 (FET 11)*.

Zambonelli F and Viroli M (2011). A survey on nature-inspired metaphors for pervasive service ecosystems. *International Journal of Pervasive Computing and Communications* **7**(3): 186–204.

Zeigler B (1984). *Multifaceted Modelling and Discrete Event Simulation*. Academic Press.

Zeigler BP (1976). *Theory of Modeling and Simulation*. John Wiley.

*Received 3 January 2012;*
*accepted 5 December 2012 after two revisions*