A Simulation Framework for Pervasive Services Ecosystems

Danilo Pianini
DEIS–Università di Bologna
via Venezia 52, 47521 Cesena, Italy
Email: danilo.pianini@unibo.it

Mirko Viroli
DEIS–Università di Bologna
via Venezia 52, 47521 Cesena, Italy
Email: mirko.viroli@unibo.it

Sara Montagna
DEIS–Università di Bologna
via Venezia 52, 47521 Cesena, Italy
Email: sara.montagna@unibo.it

***Abstract*—This paper grounds on the SAPERE project (Self-Aware PERvasive Service Ecosystems), which aims at proposing a multi-agent framework for pervasive computing, based on the idea of making each agent (service, device, human) manifest its existence in the ecosystem by a *Live Semantic Annotation* (LSA), and of coordinating agent activities by a small and fixed set of so-called *eco-laws*—sort of chemical-like reactions over patterns of LSAs. System dynamics in SAPERE is complex because of opennes and due to the self-* requirements imposed by the pervasive computing setting: a simulation framework is hence needed for what-if analysis prior to deployment. In this paper we present a prototype simulator we are developing, tested on a crowd steering scenario. Due to the role of chemical-like dynamics, this is based on a variation of an existing SSA (Stochastic Simulation Algorithm), suitable tailored to the specific features of SAPERE, including dynamicity of network topology, pattern-based application of eco-laws, and temporal triggers.**

I. Introduction and Motivation

The increasing evolution of pervasive computing is promoting the emergence of decentralised and complex infrastructures for pervasive services composed by new communication devices (e.g. mobile phones, PDA's, smart sensors, laptops). Such infrastructures include traditional services with dynamic and autonomous context adaptation (e.g., public displays showing information tailored to bystanders), as well as innovative services for better interacting with the physical world (e.g., people coordinating through their PDAs). The common languages and software infrastructures are often inadequate to face requirements of scalability, openness, adaptivity and self-organisation typical of pervasive systems. In order to better handle these scenarios, a paradigm shift towards agent world is receiving more and more attention in the scientific community. They support the realisation of distributed and eventually communicating environments where different kind of autonomous entities, the agents, are located. Agents can sense and change the environment and can interact with other agents. In particular one of the research topics about agents regards coordination, namely the way they can produce, consume and exchange information inside the pervasive system.

Different approaches were proposed in the area of coordination models and middlewares for pervasive computing scenarios: they try to account for issues related to spatiality [16], [20], spontaneous and opportunistic coordination [2], [9], self-adaptation and self-management [23]; however, most works propose ad-hoc solutions to specific problems in specific areas, and lack generality.

The SAPERE project ("Self-adaptive Pervasive Service Ecosystems") addresses the issues related to spatiality, spontaneous and opportunistic coordination, self-adaptation and self-management, in a uniform way by means of a truly self-adaptive pervasive substrate; this is a space bringing to life an ecosystem of individuals, namely, of pervasive services, devices, and humans. These are coordinated in a self-organising way by basic laws (called *eco-laws*), which evolve the population of individuals in the system, thus modelling diverse mechanisms of coordination, communication, and interaction. Technically, such eco-laws are structured as sort of chemical reactions, working on the "interface annotation" of components residing in neighbouring localities—called *LSA* (Live Semantic Annotation).

In this context models and simulation can be useful in supporting the design of pervasive systems. They give the possibility to experiment the idea of exploiting bio-inspired ecological mechanisms, showing through simulation the overall behaviour of a system designed on top of eco-laws, as well as to elaborate what-if scenarios. To capture the whole complexity of the SAPERE approach the model has to support the abstraction of *(i)* highly dynamic environment composed of different, mobile, communicating nodes and *(ii)* autonomous agents. They might be programmable through a set of chemical rules.

On one hand the adoption of the Agent-based Models (ABM) [15] seems to be quite natural as soon as the pervasive system itself is engineered adopting the agent paradigm. There are several works which apply this approach in different contexts, from social systems (see, e.g., [3]) to biological systems [18], [4]. An ABM grounds around autonomous and possibly heterogeneous agents that can be situated in an environment. They carry out the most appropriate line of action, possibly interacting with other agents as well as the environment itself. The agent behaviour is modelled through a set of rules which describe how the agent behaves according to environmental conditions. These rules can be of different types, according to the specific model / architecture: from the simple reactive rules – specifying how the agent must react to environmental stimuli or perceptions – or pro-active —specifying how the agent must behave with respect to its goals and tasks [31]. Therefore ABM

does not normally provide a way to define the behavioural rules in terms of chemical laws. In ABM the environment is also a first class abstraction whose structure, topology and dynamic can be explicitly modelled. To develop and simulate ABMs different simulation frameworks have been developed, such as MASON [14], [28], Repast [21], [26], NetLogo [30], [24], [32] and Swarm [27].

On the other hand considering pure chemical simulators with stochastic extension, such as BioPEPA [7] and BetaWB [8], helps for explicitly model the eco-laws. In this field few simulators allow to define a multi-compartment topology [1], and to the best of our knowledge no one provides facilities to move them inside an external environment. Moreover, all compartments are subject to the same set of laws, which are chemical reactions.

To take the best of both approaches we developed a brand new simulation framework, called ALCHEMIST, meant to face natively the model requirements. It implements an optimised version of the Gillespie's SSA, namely the Next Reaction Method [10], extended with the possibility to have dynamic reactions, *i.e.* reactions that can be added or removed once the simulation runs.

A notable application of the proposed approach is in crowd steering applications, in which a crowd is guided in a pervasive computing scenario depending on unforeseen events, such as the occurrence of critical events (i.e. alarms) and the dynamic formation of jams. We exemplify the approach in a crowd evacuation scenario, providing its set of eco-laws and validating it via simulation of the associated Continuos-Time Markov Chain (CTMC) model.

The remainder of this paper is organised as follows: Section II presents details about the computational model we defined and the simulator engine, Section III reports the application of the simulator in a crowd steering scenario and Section IV provides concluding remarks and discusses future works.

II. ENGINE ARCHITECTURE

In this section we first introduce how to model a chemical system in both deterministic and stochastic ways, then we show the known algorithms for stochastic simulation and our choices for a full featured high performance engine.

A. Stochastic Simulation Algorithms

A chemical system can be modelled as a single space filled with molecules that may interact through a number of reactions describing how they combine. The instantaneous speed of a reaction is called propensity and depends on the kinetic rate of the reaction and on the concentrations of all the reagents involved. For a reaction i with k reactants, j products, stoichiometric coefficients of the i-th reaction expressed as n_i and rate r of the form:

$$n_0R_0 + n_1R_1 + \ldots + n_kR_k \xrightarrow{r} m_0P_0 + m_1P_1 + \ldots m_jP_j$$

The propensity a_i is defined as in equation 1:

$$a_i = r \cdot R_0^{n_0} \cdot R_1^{n_1} \cdot \ldots \cdot R_k^{n_k} \quad (1)$$

The usual way chemical systems are studied is through differential equations describing how the concentration of each molecule continuously varies with time. In such a description the system evolution caused by a reaction is modelled through a continuous variable – the concentration – while it is an event that changes a discrete variable—the number of molecules in the system. This approximation is largely acceptable in most systems, where millions of molecules of each kind are involved simultaneously, but it's no longer accurate when only few molecules are present inside the system. In order to correctly represent this situation, a stochastic model has been proposed in [11]. The whole system is described through a single master equation that calculates the probability that at a given time t the concentration of a reactant X_i is equal to a number K_i. Solving this equation is infeasible for every non-trivial problem, but its evolution can be analysed through stochastic model checking [6], [12], running series of Monte Carlo simulations each one describing a possible path, exploiting the useful property that the probability for the simulation to run through a specific path is the probability of the path itself. This kind of description considers the whole system as a CTMC, in which the rate of the transaction representing the i-th reaction is the propensity function a_i.

In [11], two algorithms are proposed in order to correctly simulate a stochastic path of a chemical system. Those algorithms were successively improved, but every algorithm, even the optimized versions, relies on the idea that the system can be simulated by effectively executing the reactions one by one and changing the system status accordingly. Every algorithm follows four main steps:

1) select the next reaction μ to be executed;
2) calculate the time of occurrence of μ according to an exponential time distribution and make it the current simulation time;
3) change the environment status in order to reflect this execution;
4) update the propensities of the reactions.

The known techniques differ in the implementation of first and fourth steps. We will briefly present them and then justify our choice for the engine.

1) Direct Method: The direct method was first proposed in [11]. It chooses the next reaction to be executed by throwing a random number $r \leqslant \sum_i a_i$ and selecting the first reaction μ which verifies the property that $r \geqslant \sum_{i=0}^{\mu} a_i$. After the execution of μ, it updates propensities for each reaction.

2) Optimized Direct Method: The direct method can be optimised as proposed in [10] and [29] by introducing a binary search tree and a dependency graph. The former allows to choose the next reaction μ to be executed in logarithmic time, the latter to update only the propensities of those reactions in which concentration of regents is modified by the execution of μ.

3) Composition-Rejection Method: In [25] a constant time method relying on composition-rejection algorithm is proposed. The separation between the number of reactions R and the computational complexity of the algorithm is obtained by

splitting the whole set of reactions into G groups, and then arguing that G does not depend (or depends loosely) by R. It may rely on a dependency graph in order to improve the update phase.

4) First Reaction: The First Reaction Method is the dual form of the Direct Method, and was proposed first in [11]. The key idea is to calculate immediately the time of occurrence for each reaction and select the next one using the lowest time. It is demonstrably the same of the Direct Method both in soundness and in time complexity.

5) Next Reaction: The Next Reaction Method is an optimised form of the First Reaction Method first proposed in [10]. It relies on an Indexed Priority Queue (IPQ) in order to smartly sort the reactions by time, has constant time in the selection phase since the root of the IPQ is always the next reaction to execute. This algorithm requires the calculation of the times for each reaction at every update, but a dependency graph can be used, and the random re-usage is justified, speeding up consistently the times recalculation.

B. Computational Model

Before start discussing about our engine, we describe the computational model we propose in order to close the gap between the SAPERE world and the chemical simulators. In fact, these requirements will influence some aspects of the engine itself.

Our model improves the classic model of chemical reactions in three main directions, introducing the concepts of environments, nodes and neighbourhoods; extending the concept of classical chemical reaction as a set of conditions whose validity may cause the execution of a set of actions and supporting time fixed events.

First, in the classic chemical model, the environment is a single compartment that contains the molecules. This description is pretty far from the world we want to model, which is a pervasive service ecosystem. The natural extension is to consider many compartments (nodes) placed in a space (environment) which is responsible of linking them. Depending on the specific environment, nodes can be dynamically added, moved or removed. A neighbourhood is consequently a structure which contains a node "centre" and a list of all linked compartments.

Second, in classical chemical model, a reaction lists a number of reactant molecules which, combined, produce a set of product molecules. This kind of description is too strict for our purposes. A more generic concept is to consider a reaction as a set of conditions about the environment which, when matched, may allow the execution of a set of actions. A condition is a function which associates a boolean to each status of the environment, an action is a procedure which modifies it. The propensity function can no longer be simply the product of the reaction rate with the concentrations of the reactants, but needs a more generic definition too: propensity in our model is a function of the reaction rate, the conditions and the environment status.

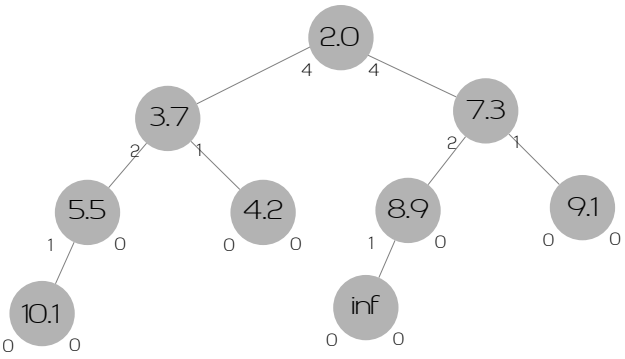

Fig. 1. Indexed Priority Queue extended with children count per branch

Third, we want to deal with events whose occurrence time does not follow an exponential law, for instance triggers, namely events which happen at a specific time regardless the previous evolution of the system. A simple example of simulation requiring triggers could be the simulation of a failure of a server in a cloud computing system: who writes the simulation has to set the failure event at a specific time, then run multiple simulations in order to understand how the system will react. Another usage of triggers appears when considering the possibility to interact with a running simulation pausing it and, exploiting triggers, interact with the environment in its current status, then resume the simulation. Even if this approach is not useful when the goal is to check the properties of a model, it could be very handy when exploring and testing it for the first times, because it allows the user to have an immediate feeling on how his model reacts to some modifications.

C. Dynamic engine

Given the model we want to simulate described in Section II-B and the algorithms presented in Section II-A, we can argue that no existing algorithm as-is is appropriate to allow our simulations. In particular, no algorithm provides facilities to add and remove reactions dynamically, and moreover it appears to be hard to inject the concept of trigger in direct method or its optimizations because they lack the possibility to choose the next reaction to execute considering immediately its time of occurrence. Our choice for the engine algorithm to extend was then restricted between the First Reaction and the Next Reaction. The latter is an optimization of the former, offers a lower computational complexity in every case and consequently can achieve higher performance. Our work had the primary goal to extend Next Reaction providing the possibility to add and remove reactions dynamically, since to the best of our knowledge no work in this sense have been ever made. In order to add this support, it is a mandatory task to provide methods to add and remove reactions from the indexed priority queue and the dependency graph.

1) Dynamic Indexed Priority Queue: A key property of the original Indexed Priority Queue proposed in [10] is that the swap procedure used to update the data structure does not

changes the balancing of the tree, ensuring optimal update times in every situation. This feature was easily achieved because no new nodes were ever added neither old ones were removed from the structure, as a consequence, once the tree is created balanced no event can occur to change its topology. This is no longer the case, and we have to provide a small extension to the structure in order to manage the balancing. Our idea is, for each node, to keep track of the number of children per branch, having in such way the possibility to keep the tree balanced when adding nodes. In figure 1 we show how the same IPQ drawn in [10] would appear with our extension. In the following algorithms, the procedure `UPDATE_AUX(n)` is the same described in [10]. Given this data structure, the procedure to add a new node `n` is the following:

```
IF root does not exists
   n is the new root
ELSE
   c <- root
   WHILE c has two children
      IF c.right < c.left
 dir <- right
      ELSE
 dir <- left
      next <- dir children
      add 1 to count of c.dir
      c <- next
   IF c has no left child
      n becomes left child of c
      set count of left nodes of c to 1
   ELSE
      n is right child of c
      set count of right nodes of c to 1
UPDATE_AUX(n)
```

The removal procedure for a node n is the following:

```
c <- root
WHILE c is not a leaf
   IF c.left > c.right
      c <- c.left
   ELSE
      c <- c.right
IF c != n
   swap c and n
   delete n
   UPDATE_AUX(c)
ELSE
   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: Since we want to support natively and efficiently the multiple compartments, we defined three contexts (also called scopes): `local`, `neighborhood` and `global`. Each reaction has an input context and an output context, meaning respectively where data influencing the rate calculus is located and and where the modifications to the environment are made.

The first issue to address is to evaluate if two reactions may influence each other, considering their contexts. We introduce a boolean procedure called `mayInfluence(r1, r2)` that operates on two reactions and returns a true value if:

- `r1` and `r2` are both on the same node OR
- `r1`'s output context is `global` OR
- `r2`'s input context is `global` OR
- `r1`'s output context and `r2`'s input context are both `neighborhood` and the node which `r1` belongs to is a neighbour of the node which `r2` belongs.

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 graph. Removing a reaction `r` requires to delete all dependencies in which `r` is involved both as influencing and influenced. Moreover, in case of change of the system topology which, a dependencies 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 which are no longer in neighbourhood.

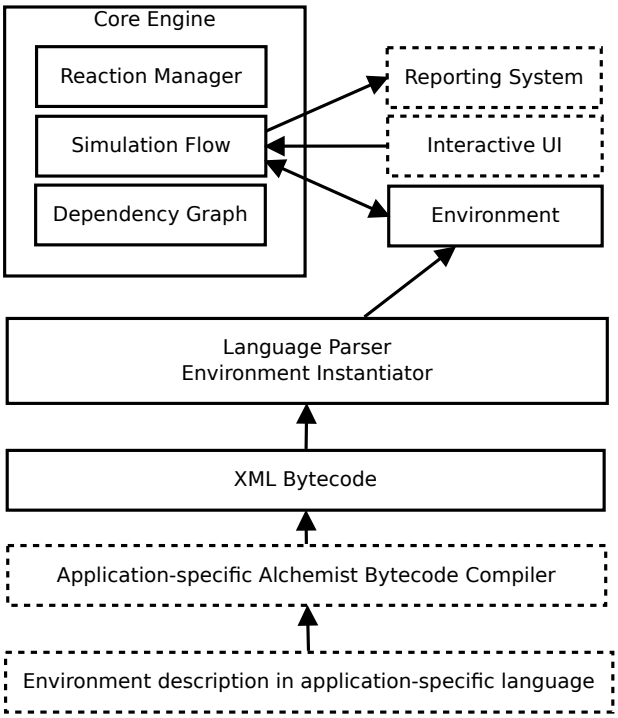

Fig. 2. ALCHEMIST architecture. Elements drawn with continuous lines indicates components common for every scenario and already developed, those with dotted lines are extension-specific components which have to be developed with the specific application in mind.

D. Engine 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 modify

without messing with engine. This modularity will make it easy to make some experiments with other engines, such as Composition-Rejection.

The framework was effectively developed using Java. Being performances a critical issue for a simulator, 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++ [5], [22]. The Java language was consequently chosen because of the excellent trade off among performances, easy portability and maintainability of the code, plus the support for concurrent programming at language level. The COLT Java library [13] provided us the mathematical function we need. In particular, it offers a fast and reliable random number generation algorithm, the so called Mersenne Twister [17].

As shown in figure 2, at the current status of development the simulations are written in a specific XML language which is interpreted in order to produce an instance of an environment. Once the environment is created, no further interpretation of the code 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 to describe environments in a machine-friendly way 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 sightly different instantiation of the model and its own language. It's up to the extensor to write a translation module from its personalised language to the ALCHEMIST XML.

III. CASE STUDY

We propose a crowd evacuation scenario as a case study. Imagine a museum with a large room, whose floor is covered with a sensor network, and an external corridor with two exits. A number of visitors are inside the main room, each one equipped with a PDA that can guide the visitor towards the exit in case of emergency. Sensors may perceive the presence of doors, fires and persons. When an emergency appears, PDAs must show the direction towards an exit, along a safe path. The system has to be resilient to changes or unpredicted situations, in particular the safe path must consider:

- *distance*: it should tend to lead to the nearest exit;
- *fire*: it should tend to stay away from fire;
- *crowd*: it should tend to avoid overcrowded paths.

A. A SAPERE model

The environment models the network of sensors. Each sensor is a node of the network. PDAs are agents dynamically linked with the nearest sensors – 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 which tend to follow the advices of the PDA. They can move of discrete steps inside the environment, but there must be a physical limit in the minimum distance between them, since two visitors can't be in the same place at the same time.

In the SAPERE flavour, all the information exchanged is in form of Live Semantic Annotations, and the rules are expressed in form of eco-laws. An LSA is simply modelled as a tuple $\langle v_1, \ldots, v_n \rangle$ (ordered sequence) of typed values, which could be for example numbers, strings or structured types. Although in the SAPERE framework LSAs are *semantic* annotations, expressing information with same expressiveness of standard frameworks like RDF, we here consider a simplified notation. There are three forms of LSAs used in this scenario:

$$\langle \texttt{source}, type, max, ann \rangle$$
$$\langle \texttt{grad}, type, value, max, ann \rangle$$
$$\langle \texttt{info}, type, value, tstamp \rangle$$

A **source** LSA is used for gradient sources: $type$ indicates the type of gradient (`fire`, `exit`, and `crowd`); max is the gradient's maximum value; and ann is the annealing factor—its purpose will be described later, along with eco-laws. A **gradient** LSA is used for individual values in a gradient: $value$ indicates the individual value; and the other parameters are like in the source LSAs. Finally, an **info** LSA is used for local values (e.g., not part of a gradient)—parameters are like in the source and gradient LSAs. The $tstamp$ reflects the time of creation of the LSA.

The sources of the gradients are injected by sensors when an exit, a fire or a number of persons is perceived, with the values $\langle \texttt{source}, \texttt{exit}, \texttt{Me}, \texttt{Ae} \rangle$ and $\langle \texttt{source}, \texttt{fire}, \texttt{Mf}, \texttt{Af} \rangle$. For the crowding information, we may assume that sensors are calibrated so as to locally inject an LSA indicating the level of crowding, *i.e.* the number of persons. The crowding LSA will look like $\langle \texttt{source}, \texttt{crowd}, \texttt{Mc}, \texttt{Ac} \rangle$ and is periodically updated by the sensor.

An eco-law is a chemical-resembling reaction working over patterns of LSAs. One such pattern P is basically an LSA which may have some variable in place of one or more arguments of a tuple, and as usual an LSA L is said to match the pattern P if there exists a substitution of variables which applied to P gives L. In Figure 3, the eco-laws for our case study are given.

As sources are established, gradients are built by the first two rules in Figure 3. The former, given a source, initiates its gradient; the latter, when a node contains a gradient LSA, spreads it to a neighbouring node with an increased value proportional to the distance between sensors indicated by the variable $\#D$. As a consequence of these laws, each node will carry a `grad` LSA indicating the topological distance from the source. When the spread values reach the maximum vale M, the gradient becomes a plateau. The spreading eco-law above may produce duplicate values in locations (due to multiple sources, multiple paths to a source, or even diffusion of multiple LSAs over time). Thus, the third eco-law retains only the minimum distance. Finally, we have to address the dynamism of the scenario where people move, fires extinguish, exits may be blocked, crowds form and dissolve. If a gradient source vanishes, the diffused values should increase (e.g., the distance to exit increases if the nearest exit is no longer available). This is the purpose of the annealing parameter in

$$\langle \texttt{source}, T, M, A\rangle \xmapsto{R_{init}} \langle \texttt{source}, T, M, A\rangle, \langle \texttt{grad}, T, 0, M, A\rangle$$

$$\langle \texttt{grad}, T, V, M, A\rangle \xmapsto{R_s} \langle \texttt{grad}, T, V, M, A\rangle, +\langle \texttt{grad}, T, \texttt{min}(V{+}\#D, M), M, A\rangle$$

$$\langle \texttt{grad}, T, V, M, A\rangle, \langle \texttt{grad}, T, W, M, A\rangle \rightarrow \langle \texttt{grad}, T, \texttt{min}(V, W), M, A\rangle$$

$$\langle \texttt{grad}, T, V, M, A\rangle \xmapsto{R_{ann}(A)} \langle \texttt{grad}, T, V{+}1, M, A\rangle$$

$$\begin{array}{r} \langle \texttt{grad}, \texttt{exit}, E, Me, Ae\rangle, \\ \langle \texttt{grad}, \texttt{fire}, F, Mf, Af\rangle, \\ \langle \texttt{info}, \texttt{crowd}, CR, TS\rangle \end{array} \xmapsto{R_{att}} \begin{array}{l} \langle \texttt{grad}, \texttt{exit}, E, Me, Ae\rangle, \langle \texttt{grad}, \texttt{fire}, F, Mf, Af\rangle, \\ \langle \texttt{info}, \texttt{crowd}, CR, TS\rangle, \\ \langle \texttt{info}, \texttt{attr}, (Me - E)/(1 + (Mf - F) + k \times (Mc - C)), \#T\rangle \end{array}$$

$$\langle \texttt{info}, \texttt{attr}, A, TS\rangle, \langle \texttt{info}, \texttt{attr}, A2, TS{+}T\rangle \rightarrow \langle \texttt{info}, \texttt{attr}, A2, TS{+}T\rangle$$

$$\begin{array}{r} \langle \texttt{info}, \texttt{escape}, L\rangle, \langle \texttt{info}, \texttt{attr}, A, TS\rangle, \\ +\langle \texttt{info}, \texttt{attr}, A{+}\Delta, TS2\rangle \end{array} \xmapsto{R_{disp}(\Delta)} \begin{array}{l} \langle \texttt{info}, \texttt{escape}, \#O\rangle, \langle \texttt{info}, \texttt{attr}, A, TS\rangle, \\ +\langle \texttt{info}, \texttt{attr}, A + \Delta, TS2\rangle \end{array}$$

Fig. 3. Eco laws.

```
<?xml version="1.0" encoding="UTF8"?>
<environment ... >
  <concentration type="DoubleConcentration"></concentration>
  <position type="Continuous2DEuclidean"></position>
  <molecule name="exitSource" type="Molecule" p0="exitSource"></molecule>
  <molecule name="exitGrad" type="Molecule" p0="exitGrad"></molecule>
  ...
  <node name="door1" ... >
    <content exitSource="1"></content>
    <reaction name="pumpToField" type="ExpTimeReaction" p0="n0" p1="1">
      <condition type="MoleculePresentCondition" p0="exitSource" p1="door1"
                                                          p2="1"></condition>
      <action name="setGrad" type="SetLocalMoleculeConcentration" p0="door1"
                                              p1="exitGrad" p2="0"></action>
    </reaction>
    ...
  </node>
  ...
</environment>
```

Fig. 4. Alchemist XML code snipped of the case study, with the translation of the first eco-law.

the gradient LSAs: it defines the rate of fourth eco-law, which continuously tends to level up gradient values, encouraging the replacement of old values by more current ones. The R_{ann} rate is directly proportional to A. When a fire is put out, for example, this eco-law will gradually raise the fire gradient to the point where it reaches the maximum, indicating no fire. Annealing may introduce a burden on the system, therefore high annealing values should only be used for gradients that have to change often or quickly.

Based on exit distance, fire distance and crowding, a location can be ranked as more or less "attractive" to be part of an escape path. This is done via an *attractiveness* value automatically attached to each node by fifth eco-law. Coefficient k (tuned by simulation) is used to weight the effect on crowding on attractiveness. As gradients evolve, older attractiveness LSAs are replaced with newer ones with seventh eco-law (T is assumed positive).

Each location contains by default an LSA of the form $\langle \texttt{info}, \texttt{escape}, L, TS\rangle$, where L is the direction to be suggested by the PDA. In principle, the neighbour with the highest attractiveness should be chosen, but a more resilient solution is to tie the markovian rate of eco-laws to the attractiveness of neighbours, so that the highest probability is to point the best neighbour, with a possibility to point a less-than-optimal (but

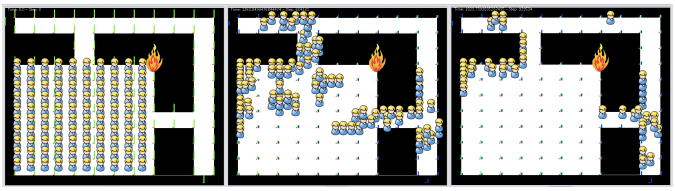

Fig. 5. A simulation run of the reference exposition: three snapshots of the ALCHEMIST graphic reporting module with this simulation

still attractive) neighbour, as described by the last eco-law. The rate is proportional to the difference in attractiveness between the node and its neighbour (Δ). The higher the Δ, the higher the rate. Note that Δ is a positive value, hence $A + \Delta$ implies that the last eco-law only considers neighbours with a higher attractiveness, i.e., the PDA will not point *away* from the exit.

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

More details about the model of the case study are given in [19].

B. Simulator configuration

The behaviour of each agent is programmed according to the eco-laws coordination model explained in Figure 3. We here present simulations conducted over an exposition structured as shown in Figure 5, where three snapshots of a simulation run are reported: all the people in the room start moving towards one of the two exists (located at the ends of the corridor) because of the fire in the top-right corner of the room. Note in third snapshot that a person is walking in the middle of the corridor, for she was suggested to go to a farther exit because of the corridor jam at the bottom-right. Rooms and corridors are covered by a grid of locations hosting sensors, one per meter in the room, one per two meters in the corridor: such locations are the infrastructure nodes where LSAs are reified. The maximum values for the gradients are set to: $\texttt{Me} = 30$, $\texttt{Mf} = 3$, $\texttt{Mc} = 20$. The PDA of each person is modelled as a mobile node, able to perceive the attractiveness gradient in the nearest sensor locations: accordingly, the person moves in the suggested direction.

Each eco-law in Figure 3 is modeled inside the simulator as a reaction. The behaviour of visitors is a reaction too, featuring a special action in which the behaviour of the visitors is expressed.

Through this scenario many innovative aspects of ALCHEMIST can be stressed: we have mobile compartments, a triggers (which are exploited in order to set up fire), dynamically changing neighbourhoods and a pretty rich scenario with up to 316 nodes moving inside and exiting an environment with physical obstacles.

C. Parameter tuning and simulation results

ALCHEMIST offers full support to run concurrently multiple simulations in order to tune parameters. In this scenario this

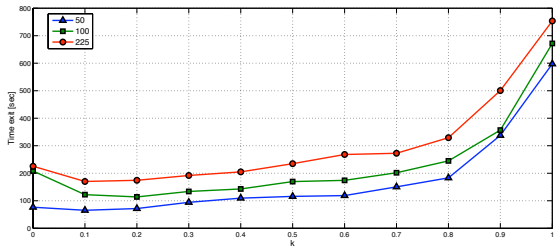

Fig. 6. Results of the k parameter analysis.

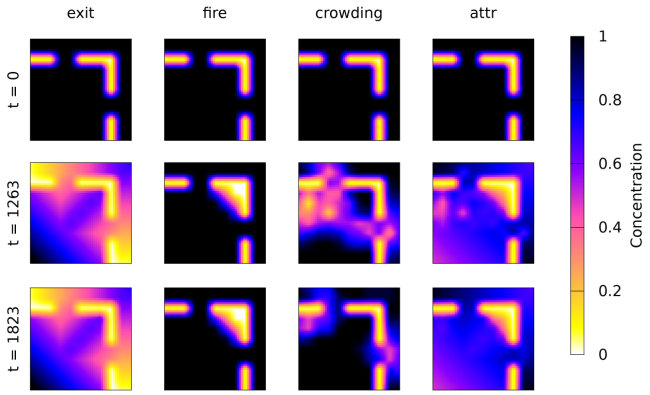

Fig. 7. Gradients for the snaphots in Figure 5. Concentration is the gradient value normalised to its maximum value.

feature is exploited to optimize the parameter k, finding out how to fix it in the fifth reaction of Figure 3 in order to grant a low exit time in different cases. We tested it with 50, 100 and 225 visitors. Results are shown in Figure 6.

Figure 7 shows the gradients of exit, fire, crowding and attractiveness (one per column) corresponding to the simulation steps of Figure 5 (one step per row). At $t = 0$, gradients are level; with time, the gradient self-modify—it is easy to see the exits, fire and crowds in the respective gradients. The crowding gradient in the third column changes dynamically during simulation according to the movement of people. The last column shows the attractiveness gradient, computed from the other three gradients. Note how the second snapshot shows an attractiveness "hole" in the middle of the room and in the corridor due to crowding.

IV. CONCLUSION

In the SAPERE metaphor, the ideal level of abstraction to reach in order to easily and correctly model and simulate pervasive systems stands between the ABM and biochemistry-oriented simulators. In this work we shown the ALCHEMIST simulation framework, meant to fully support this way to think pervasive systems. This framework embraces the SAPERE vision and allows to approach the simulation of agent systems in a new flavour, describing the system in terms of reaction-like laws and having consequently the possibility to rely on all the work already made about CTMC. We shown a case study

whose complexity overcomes the expressiveness possibility of classical biochemistry-oriented simulation frameworks, and we analysed it exploiting the same CTMC mathematical support. Perspectives for the immediate future include a comparison in terms of performance and expressiveness with the ABM simulation frameworks, such as Repast and NetLogo, and the analysis, modelling and simulation of further scenarios, with different types of complexity so to stress the potentialities of ALCHEMIST. Future work are also devoted to theoretically compare the CTMC model with the Discrete Event Simulation approach at the simulation, normally adopted in the ABM simulators.

ACKNOWLEDGMENT

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

REFERENCES

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

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

[3] S. Bandini, S. Manzoni, and G. Vizzari. Crowd Behavior Modeling: From Cellular Automata to Multi-Agent Systems. In A. M. Uhrmacher and D. Weyns, editors, *Multi-Agent Systems: Simulation and Applications*, Computational Analysis, Synthesis, and Design of Dynamic Systems, chapter 13, pages 389–418. CRC Press, June 2009.

[4] G. Beurier, F. Michel, and J. Ferber. A morphogenesis model for multiagent embryogeny. In L. M. Rocha, L. S. Yaeger, M. A. Bedau, D. Floreano, R. L. Goldstone, and A. Vespignani, editors, *Artificial Life X: Proceedings of the Tenth International Conference on the Simulation and Synthesis of Living Systems*, pages 84–90. MIT Press, Cambridge, MA, 2006.

[5] J. M. Bull, L. A. Smith, C. Ball, L. Pottage, and R. Freeman. Benchmarking java against c and fortran for scientific applications. *Concurrency and Computation: Practice and Experience*, 15(3-5):417–430, 2003.

[6] M. Casadei and M. Viroli. A framework to specify and verify computational fields for pervasive systems. *ACM Transactions on Autonomous and Adaptive Systems*, 2010.

[7] F. Ciocchetta and M. L. Guerriero. Modelling biological compartments in Bio-PEPA. *Electronic Notes in Theoretical Computer Science*, 227:77–95, 2009.

[8] L. Dematté, C. Priami, A. Romanel, and O. Soyer. Evolving blenx programs to simulate the evolution of biological networks. *Theoretical Computer Science*, 408(1):83–96, 2008.

[9] C.-L. Fok, G.-C. Roman, and C. Lu. Enhanced coordination in sensor networks through flexible service provisioning. In J. Field and V. T. Vasconcelos, editors, *Proceedings of COORDINATION 2009*, volume 5521 of *LNCS*, pages 66–85. Springer-Verlag, 2009.

[10] M. A. Gibson and J. Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *J. Phys. Chem. A*, 104:1876–1889, 2000.

[11] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, December 1977.

[12] T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet. Approximate probabilistic model checking. In *VMCAI*, pages 73–84, 2004.

[13] W. Hoschek. *The Colt Distribution: Open Source Libraries for High Performance Scientific and Technical Computing in Java*. Geneva, 2004.

[14] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. C. Balan. Mason: A multiagent simulation environment. *Simulation*, 81(7):517–527, 2005.

[15] C. M. Macal and M. J. North. Tutorial on agent-based modelling and simulation. *Journal of Simulation*, 4:151–162, 2010.

[16] M. Mamei and F. Zambonelli. Programming pervasive and mobile computing applications: The tota approach. *ACM Trans. Softw. Eng. Methodol.*, 18(4):1–56, 2009.

[17] M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, 1998.

[18] S. Montagna, N. Donati, and A. Omicini. An agent-based model for the pattern formation in Drosophila Melanogaster. In H. Fellermann, M. Dörr, M. M. Hanczyc, L. Ladegaard Laursen, S. Maurer, D. Merkle, P.-A. Monnard, K. Stoy, and S. Rasmussen, editors, *Artificial Life XII*, chapter 21, pages 110–117. The MIT Press, Cambridge, MA, USA, 2010. Proceedings of the 12th International Conference on the Synthesis and Simulation of Living Systems, 19-23 Aug. 2010, Odense, Denmark.

[19] S. Montagna, M. Viroli, M. Risoldi, D. Pianini, and G. Di Marzo Serugendo. Self-organising pervasive ecosystems: A crowd evacuation example. 29-30 September 2011. Submitted at the 3rd International Workshop on Software Engineering for Resilient Systems.

[20] A. L. Murphy, G. P. Picco, and G.-C. Roman. Lime: A model and middleware supporting mobility of hosts and agents. *ACM Trans. on Software Engineering and Methodology*, 15(3):279–328, 2006.

[21] M. J. North, T. R. Howe, N. T. Collier, and J. R. Vos. A declarative model assembly infrastructure for verification and validation. In S. Takahashi, D. Sallach, and J. Rouchier, editors, *Advancing Social Simulation: The First World Congress*, pages 129–140. Springer Japan, 2007.

[22] B. Oancea, I. G. Rosca, T. Andrei, and A. I. Iacob. Evaluating java performance for linear algebra numerical computations. *Procedia CS*, 3:474–478, 2011.

[23] P. V. Roy, S. Haridi, A. Reinefeld, J.-B. Stefany, R. Yap, and T. Coupaye. Self-management for large-scale distributed systems: an overview of the selfman project. In *Formal Methods for Components and Objects, LNCS No. 5382*, pages 153–178. Springer Verlag, 2008.

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

[25] A. Slepoy, A. P. Thompson, and S. J. Plimpton. A constant-time kinetic monte carlo algorithm for simulation of large biochemical reaction networks. *The Journal of Chemical Physics*, 128(20):205101, 2008.

[26] R. D. Team. http://repast.sourceforge.net/. Repast home page.

[27] S. D. Team. http://www.swarm.org/index.php/Main_Page. Swarm home page.

[28] G. M. University. http://www.cs.gmu.edu/\~eclab/projects/mason/. MASON home page.

[29] C. Versari and N. Busi. Efficient stochastic simulation of biological systems with multiple variable volumes. *Electr. Notes Theor. Comput. Sci.*, 194(3):165–180, 2008.

[30] U. Wilensky and CCL. http://ccl.northwestern.edu/netlogo/index.shtml. NetLogo home page.

[31] M. Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, 1st edition, June 2002.

[32] S. Yildirim, G. L. Dam, and J. C. Houk. The mind agents in netlogo 3.1. In *SpringSim (2)*, pages 137–143, 2007.