# Statistical analysis of chemical computational systems with MULTIVESTA and ALCHEMIST

***Abstract*—The chemical-oriented approach is an emerging paradigm for programming the behaviour of densely distributed and context-aware devices (e.g. in ecosystems of displays tailored to crowd steering, or to obtain profile-based coordinated visualization). Typically, the evolution of such systems cannot be easily predicted, thus making of paramount importance the availability of techniques and tools supporting prior-to-deployment analysis. Exact analysis techniques do not scale well when the complexity of systems grows: as a consequence, approximated techniques based on simulation assumed a relevant role. This work presents a new simulation-based distributed tool addressing the statistical analysis of such a kind of systems, which has been obtained by chaining two existing tools: MULTIVESTA and ALCHEMIST. The former is a recently proposed lightweight tool which allows to enrich existing discrete event simulators with distributed statistical analysis capabilities, while the latter is an efficient simulator for chemical-oriented computational systems. The tool is validated against a crowd steering scenario, and insights on the performance are provided by discussing how these scale distributing the analysis tasks on a multi-core architecture.**

## I. INTRODUCTION

The number of intercommunicating devices spread around the world is constantly increasing: sensors, phones, tablets, eyeglasses and many other everyday objects are carrying more and more computational and communicational capabilities. Such a computationally dense environment called for new programming approaches, many of them inspired by natural systems, e.g. biological [1], [2], physical [3] and chemical [4], [5]. In all of them, the overall system's behaviour emerges from local, simple and probabilistic interactions among the devices composing the computational continuum. For this reason, most of the work in literature focuses on modelling single devices. When this reductionist point of view is adopted, a difficult task in the development methodology is to assert system properties: the system's evolution can not be easily predicted and thus multiple simulations runs are performed [6]–[9]. Obvious questions accompany these procedures: *how reliable are the obtained values? How is the number of performed simulation chosen?* And *how many simulations are required in order to state system properties with a certain degree of confidence?* Moreover, there is frequently a lack of decoupling between the model specification and the definition of the system's properties of interest: they are often embedded in the model, and their values are obtained via logging during the simulation process.

This work presents a new tool obtained chaining ALCHEMIST [10], an efficient state-of-the-art simulator for chemical-oriented computational systems, with MULTIVESTA [11], a recently proposed lightweight tool which allows to enrich existing discrete event simulators with distributed statistical analysis capabilities. The result is thus a statistical analysis tool tailored to chemical-inspired pervasive systems. To sum up, the simulator has been enriched with: `(1)` a language (MULTIQUATEX) to compactly and cleanly express systems properties, decoupled from the model specification; `(2)` the automatized estimation of the expected values of MULTIQUATEX expressions with respect to $n$ independent simulations, with $n$ large enough to respect a user-specified confidence interval; `(3)` the generation of gnuplot input files to visualize the obtained results; `(4)` a client-server architecture to distribute simulations. The tool is validated by analyzing a crowd steering model reminiscent of the one presented in [12].

**Synopsis.** §II introduces ALCHEMIST and describes the crowd steering scenario, while §III outlines the main features of MULTIVESTA. Then §IV discusses the integration of the two tools, while §V validates the obtained tool. Finally, §VI reports some concluding remarks and future works.

## II. ALCHEMIST

ALCHEMIST is a simulator targeting chemical-oriented computational systems. It is aimed at bridging the gap between: `(1)` tools that provide programming/specification languages devoted to ease the construction of the simulation process, especially targeting computing and social simulation (e.g. as in the case of simulation based on multi-agents [8], [13]–[16]); and `(2)` tools that stick to foundational languages, which typically offer better performance, mostly used in biology-oriented applications [17]–[20]. ALCHEMIST extends the basic computational model of chemical reactions – still retaining its high performance – aiming at easing its applicability to complex situated computational systems (following the chemical-oriented abstractions studied in [4]). In particular, ALCHEMIST is based on an optimised version of the Gillespie's SSA [21] called Next Reaction Method [22], properly extended with 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 as generic as possible, and as such they can have a wide range of applications like pervasive computing, social interactions and computational biology [23].

A detailed description of the simulator's meta-model and its engine's internals are out of the scope of this work. The interested reader can find a deeper insight in [10]. ALCHEMIST is written in Java and is still actively developed. It currently consists of about 630 classes for about 100'000 lines of code, and it is released [1] as open source (GPL licensed).

### A. A crowd steering scenario

Our reference scenario, depicted in Figure 1, consists of a hall of $20 \times 10$ meters on whose floor a regular grid of

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

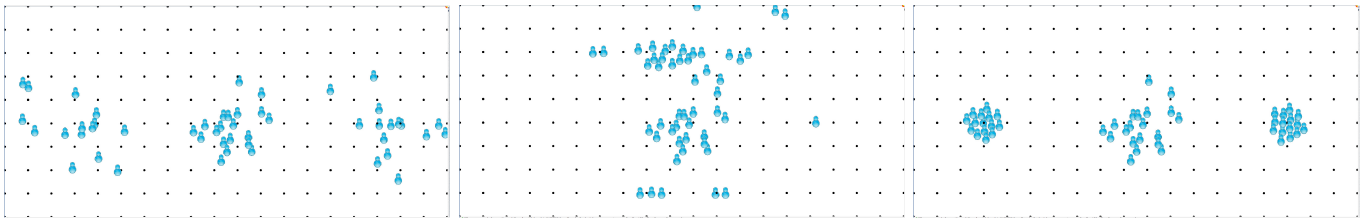

Fig. 1. Three snapshots of the reference scenario showing: the system at the initial stage (left), an intermediate state (centre), and the final situation (right).

200 computationally enabled sensors is deployed. In the hall, three groups of 15 people each are located. One group, on the left, wants to get to a point of interest (POI) on the right. Another group, on the right, desires to get to a POI on the left. Finally, a third group stands in the centre of the hall. Each person is equipped with a smartphone. Devices (i.e. sensors and smartphones) are able to communicate with devices within their communication range (1.5 meters). Sensors can count the people in such range, while smartphones provide real time suggestions to the user about the direction to be taken. Users are supposed to follow the advices of their smartphone whenever possible, however, a pedestrian model that makes them subject to physical interactions is used. Such model also includes the social desire of not breaking the group, which may lead the users to choose a direction different from the suggested one.

In such a scenario, we want to deploy a fully distributed crowd steering system, in which the sensors automatically build optimal paths, so that from whichever location in the hall, each user can move towards the desired direction. We also want to employ the local real time information about the number of users in the surroundings in order to make less crowded paths advantaged over those who would steer the user on a jammed area. Finally, we want to detect the user preference about the destination, and correctly select which route to follow.

*1) A model based on computational fields:* Our strategy is to rely on computational fields, namely a distributed data structure that carries, for each point in space, a contextualized information. A very notable example of computational field is the gradient, in which each device is in charge to estimate its distance from the closest device designated as *source of the gradient* (e.g. the POIs). Figure 2 shows intuitively how the spatial gradient structure works. If we consider each of the two POIs of our scenario as a *source* of a different gradient, and we program our sensors properly, we will have each device instructed on its distance from each of the POIs. Along with the distance information, also the next hop towards each of the POIs can be stored. Consequently, if descended, the gradient guides the user towards its POI along an optimal path. This strategy has been used before in crowd simulations, with the gradient statically computed at the beginning of the simulation in order to obtain information about the scenario and the obstacles [24].

For our purposes, however, a static gradient is not enough. In fact, we want to dynamically modify this spatial data structure in order to take into account the crowding level of each area: this means that each sensor must dynamically manipulate its local information, and propagate it coherently. Even if not considered in our reference scenario, similar requirements arise in case we deal with unpredicted events, such as network nodes failures, addition or movement. In order to obtain such dynamicity, gradients must then provide a feature commonly identified as *self-healing* ability. Fortunately, nice algorithms have already been proposed for self-healing gradients [25], [26]. Similar manipulations of self-healing gradients have been used in other works, by aggregating multiple gradients [27], by combining them with local information [28], or both [29].

*2) The pedestrians model:* ALCHEMIST provides a realistic pedestrian model, based primarily on [30]. We relied on such model to obtain a realistic physical interaction among people. A deep analysis of this model is out of the scope of this work.

*3) Implementation with a chemical metaphor:* In this scenario we propose a solution relying on the SAPERE concepts of "live semantic annotation" (LSA) and "eco-law" [4]. LSAs are "semantic annotations" in the sense that they can carry semantic information, and they are "live" in the sense that they are in charge of reflecting the perceived status of the world for every device. All the data in the system are encoded as LSA, for instance in the considered scenario both the spatial gradient and the crowd level detection are reified in form of LSAs. The eco-laws are a particular class of rewriting rules that manipulate, aggregate, create and delete LSAs.

```
environment 0 1.5                                                          1
lsa source <source, Type, Distance>                                        2
lsa source_template <source, Type, Distance>                               3
lsa gradient_template <grad, Type, Distance>                               4
lsa gradient1 <grad, target, Distance>                                     5
lsa gradient2 <grad, target2, Distance>                                    6
lsa crowd <crowd, L>                                                       7
/********************** Sensors **********************/                  8
place 200 nodes in rect (0,0,19,9) interval 1                              9
containing                                                                 10
in point (16, 4) <source, target, 0>                                       11
in point ( 3, 4) <source, target2, 0>                                      12
with reactions                                                             13
reaction SAPEREGradient params "ENV,NODE,RANDOM,                           14
    source_template,gradient_template,2,((Distance+#D)
    +(0.5*L)),crowd,10000,10" []-->[]
eco-law compute_crowd []-1-> [agent CrowdSensor params "ENV,               15
    NODE"]
/*************** Group on the left side ***************/                 16
place 15 nodes in circle (3, 4, 3)                                         17
containing in all <person> <group1>                                        18
with reactions                                                             19
[]-100->[agent SocialForceEuropeanAgent params "ENV,NODE,                  20
    RANDOM,gradient1,2,1,false"]
/*************** Group on the right side ***************/                21
place 15 nodes in circle (16, 4, 3)                                        22
containing in all <person> <group2>                                        23
with reactions                                                             24
[]-100->[agent SocialForceEuropeanAgent params "ENV,NODE,                  25
    RANDOM,gradient2,2,2,false"]
/**** Group standing still in the center of the hall ****/                26
place 20 nodes in circle (10, 4, 2)                                        27
containing in all <person>                                                 28
```

Listing 1. SAPERE-DSL Specification for our experiment

The details on the implementation of the crowd steering algorithm are not reported in this paper, but are available in

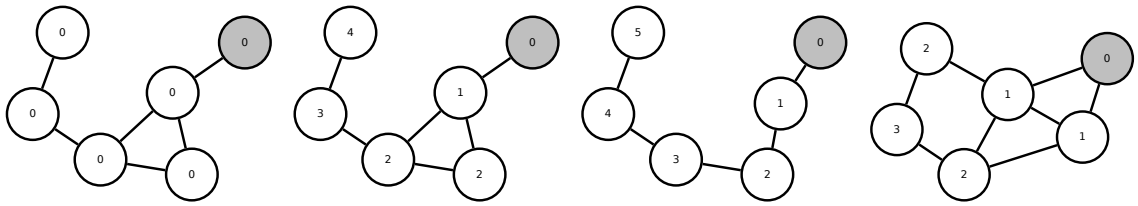


Fig. 2. A spatial gradient in a simple network in which the nodes are at distance 1 from their neighbours. The grey node is the source, and nodes are labelled with the known distance from the source. From left: `(i)` the initial status of the network; `(ii)` the network once the gradient stabilised `(iii)` the re-shaping of the gradient due to a node movement and subsequent link breaking; `(iv)` the re-shaping of the gradient due to node movement and subsequent new link creation

[12]. For the sake of reproducibility, in Listing 1 we show the code snippet used to model the scenario in ALCHEMIST.

## III. MULTIVESTA

MULTIVESTA [2] is a recently proposed statistical analyzer for probabilistic systems [11], extending VESTA [31] and PVESTA [32]. The analysis algorithms of MULTIVESTA are independent of the used model specification language: it is only assumed that discrete event simulations can be performed on the input model. As described in [11], the tool offers a clean interface to integrate existing discrete event simulators, enriching them with a property specification language, and with efficient distributed statistical analysis capabilities.

MULTIVESTA performs a statistical (Monte Carlo based) evaluation of MULTIQUATEX expressions, allowing to query about expected values of observations performed on simulations of probabilistic models. A MULTIQUATEX expression may regard more than a measure of a model, in which case the same simulations are reused to estimate them, thus improving the performance of the analysis tasks. Moreover, the tool has a client-server architecture allowing to distribute the simulations on different machines. A detailed description of MULTIQUATEX and of the procedure to estimate its expressions, omitted in this work due to space constraints, is given in [11], [33]. The tool also supports the transient fragment of probabilistic computation tree logic (PCTL) [34] and continuous stochastic logic (CSL) [35], [36], for which statistical model checking algorithms based on the invocation of a series of inter-dependent statistical hypothesis testing are implemented [37]. However, this work focuses on MULTIQUATEX, as it generalizes the two mentioned logics [33].

Before defining a MULTIQUATEX expression, it is necessary to specify the state characteristics to be observed. This model-specific step "connects" MULTIQUATEX with the simulated model. In particular, the state observations are offered via the `rval(`$i$`)` predicate which returns a number in the real domain for each observation $i$. As sketched in Section IV, for the crowd steering scenario `s.rval(0)` corresponds to the current simulated time, `s.rval(3)` counts the number of LSAs in the system, while `s.rval(11)`, `s.rval(12)` and `s.rval(13)` count, respectively, the number of people that have reached the POI on the right, the one on the left, and both. Finally, `s.rval(14)` returns the average connectivity degree of the devices (i.e. sensors and smartphones).

A MULTIQUATEX expression consists of a set of definitions of *parametric recursive temporal operators*, followed by a list of

[2]http://code.google.com/p/multivesta/.

```
time@POI() = if{s.rval(13) == 30.0} then s.rval(0)                    1
                                    else #time@POI() fi;              2
eval E[ time@POI() ] ;                                                3
```

Listing 2. The simple MULTIQUATEX expression $Q_1$

*`eval` clauses*. Each `eval` clause relies on the defined temporal operators, and specifies a system property whose expected value must be evaluated. As an example, Listing 2 depicts the simple MULTIQUATEX expression $Q_1$ which intuitively reads as: "*compute the expected value of the time necessary to let all the individuals reach their target*". More in particular, lines `1-2` define a recursive temporal operator, composed by the name of the operator (`time@POI()`), and by a *path expression* representing its body, i.e. a real-typed predicate possibly evaluated after performing steps of simulation. Line `3` provides one *`eval` clause*, specifying that we are interested in evaluating the expected value of `time@POI()`.

In order to evaluate a MULTIQUATEX expression, MULTIVESTA performs several simulations, obtaining from each a list of samples (real numbers). One sample for each `eval` clause is obtained, thus all the queried measures are evaluated using the same simulations, improving performance. When evaluating the samples of a simulation, `s` is associated to the initial state of the system, and then the `eval` clauses are evaluated. Consider the simple case of $Q_1$, having just one `eval` clause: at the first step of the simulation the guard of the `if` statement (`s.rval(13) == 30`) is evaluated: "*does in* `s` *all the* 30 *individuals have reached their target?*". If the guard is evaluated to true, then `s.rval(0)` is returned, i.e. the current simulated time. Otherwise the expression is evaluated as `#time@POI()`: MULTIVESTA orders the simulator to advance of one step, updates `s`, and then recursively evaluates `time@POI()`. Note that the operator # (named next) triggers the execution of a step of simulation, thus if it is used in recursive temporal operators (like in $Q_1$), it allows to query properties of states obtained after an unspecified number of steps. The evaluation evolves as described until a state satisfying the guard of the `if` statement is reached (which always happens in the considered model).

The case of expressions with more `eval` clauses is similar, the only difference is that, at each step of the simulation, all the `eval` clauses are evaluated: for each of them, either a real value, or the # operator followed by a temporal operator is returned. In the first case, the sample relative to the `eval` clause is obtained, and thus the `eval` clause is ignored for the rest of the simulation, in the second case a step of simulation is required. The evaluation of the samples in a simulation terminates when all the `eval` clauses completed their evaluation.

```
people@POI(x) = if{s.rval(0) >= x} then s.rval(13)
                                   else #people@POI(x) fi;
eval E[ people@POI(30.0) ] ;
```

Listing 3. The MULTIQUATEX expression $Q_2$

Basing on the samples obtained from simulations, MULTIVESTA estimates the expected values of MULTIQUATEX expressions with respect to two user-defined parameters: $\alpha$ and $\delta$. Considering the case of simple expressions with just one `eval` clause, the estimations are computed as the mean value of the $n$ samples obtained from $n$ simulations, with $n$ large enough to grant that the size of the $(1-\alpha)*100\%$ *Confidence Interval* (CI) is bounded by $\delta$. In other words, if a simple MULTIQUATEX expression is estimated as $\overline{x}$, then, with probability $(1-\alpha)$, its actual expected value belongs to the interval $[\overline{x}-\delta/2, \overline{x}+\delta/2]$.

Again, the case of expressions with multiple `eval` clauses is similar. Note that the `eval` clauses may regard values of different orders of magnitude, and thus the user may provide a list of $\delta$ rather than just one. After having obtained a sample for every `eval` clause from a simulation, these values are used to update the means of the samples obtained from previous simulations (one mean per `eval` clause). If the CIs have been reached for every `eval` clause, the evaluation of the expression is terminated, otherwise further simulations are performed. Note that each `eval` clause may require a different number of simulations to reach the required CI. Once the CI of an `eval` clause has been reached, such `eval` clause is ignored in eventual further simulations performed for other `eval` clauses.

Another interesting expression is $Q_2$ of Listing 3, reading: *compute the expected number of people reaching the target after* 30 *units of simulated time*. $Q_2$ shows that temporal operators can have parameters (variables). Variables have to be bounded, i.e. if they are in the right-hand-side of a temporal operator definition (i.e. after the equals sign), then they also have to be in its left-hand-side, so that a value can be assigned to them. Noteworthy, if `rval(13)` would evaluate to 1.0 in case a certain event happens in a simulation, and to 0.0 otherwise, then $Q_2$ would estimate the probability of such event.

The simple expressions $Q_1$ and $Q_2$ query a measure of the system (i.e. `rval(0)` in $Q_1$, or `rval(13)` in $Q_2$), while one may be interested in more. As said, MULTIQUATEX expressions may have lists of `eval` clauses, each studying a different measure. We refer to expressions having more `eval` clauses as *multi-expressions*. Intuitively, a multi-expression with $n$ `eval` clauses corresponds to $n$ expressions sharing the same temporal operators but having each one of the `eval` clauses. However, the multi-expression is more compact and is evaluated performing less simulations: just the maximal number of simulations required by the single simple expressions. Listing 4 provides the multi-expression $MQ_{1-2}$. It is not difficult to see that $MQ_{1-2}$ corresponds to $Q_1$ and $Q_2$.

```
time@POI() = if{s.rval(13) == 30.0} then s.rval(0)
                                    else #time@POI() fi;
people@POI(x) = if{s.rval(0) >= x} then s.rval(13)
                                   else #people@POI(x) fi;
eval E[ time@POI() ] ; eval E[ people@POI(30.0) ] ;
```

Listing 4. The MULTIQUATEX expression $MQ_{1-2}$

```
people@POI(x) = if{s.rval(0) >= x} then s.rval(13)                    1
                                   else #people@POI(x) fi;            2
eval E[ people@POI(10.0) ] ; eval E[ people@POI(15.0) ] ;             3
eval E[ people@POI(20.0) ] ; eval E[ people@POI(25.0) ] ;             4
eval E[ people@POI(30.0) ] ; eval E[ people@POI(35.0) ] ;             5
```

Listing 5. The MULTIQUATEX expression $MQ_2$

```
people@POI(x) = if{s.rval(0) >= x} then s.rval(13)                    1
                                   else #people@POI(x) fi;            2
eval parametric(E[ people@POI(x) ],x,10.0,5.0,35.0) ;                 3
```

Listing 6. $MQ_2$ as a parametric multi-expression

Listing 5 provides another interesting multi-expression ($MQ_2$), that estimates the excepted number of people reaching their target at the varying of the simulated time (i.e. at time 10, 15, 20, 25, 30 and 35). In order to ease the writing of multi-expressions like $MQ_2$, MULTIQUATEX provides *parametric multi-expression* (or *parametric expression* in short), i.e. some syntactic sugar (a macro) that allows to concisely write multi-expressions evaluated at the varying of a parameter. Listing 6 depicts a parametric expression corresponding to $MQ_2$. In line 3, the keyword `parametric` is used: provided a path expression (`people@POI(x)`), a variable ($x$) and a range of values specified as min (10.0), increment (5.0) and max (35.0), the keyword is unrolled in the corresponding list of `eval` clauses (in this case those of Listing 5). Noteworthy, `parametric` takes as first parameter a list of path expressions, allowing to study more measures at the varying of a parameter.

It is assumed that expressions are properly typed: the guards of `if` statements must be booleans, while the path expressions in the `eval` clauses must be real. Moreover, we restrict to *bounded expressions*, i.e. the subset of expressions which can be evaluated performing a finite number of steps of simulation.

## IV. INTEGRATING MULTIVESTA AND ALCHEMIST

This section describes the integration of MULTIVESTA and ALCHEMIST. Some steps (Section IV-A), have been tackled once and for all, while others are model-specific, and are thus related to the crowd steering scenario (Section IV-B).

### A. Simulator-specific integration

Essentially, in order to allow the interaction with MULTIVESTA, ALCHEMIST has to fulfill two requirements: (1) the ability to advance the simulation in a step-by-step manner (which is provided by the `playSingleStepAndWait` method in `ISimulation` interface); (2) the ability to analyse the model status after each simulation step, providing measures in form of real numbers about properties of interest.

Since both MULTIVESTA and ALCHEMIST are Java-based, their interaction has been easily realized by subclassing the `NewState` class of MULTIVESTA. The obtained `AlchemistState` class is sketched in Listing 7, where unnecessary details are omitted. The new class contains some ALCHEMIST-specific code, providing MULTIVESTA with the simulation control and proper entry points for the analysis.

```
1  public class AlchemistState<N extends Number, D extends
       Number, T> extends NewState {
2    private final EnvironmentBuilder<N, D, T> eb;
3    private final long maxS;
4    private final ITime maxT;
5    private ISimulation<N, D, T> sim;
6    ...
7    public AlchemistState(final ParametersForState params)
         throws ...{
8      super(params);
9      final StringTokenizer otherparams = new StringTokenizer(
           params.getOtherParameters());
10   // Initialization of Alchemist-specific parameters and
       execution environment resorting to otherParams
11   }
12   ...
13   public void setSimulatorForNewSimulation(final int seed) {
14     /* Stop current simulation, create a new one. */
15     ...
16     sim.stop();
17     sim.waitForCompletion();
18     ...
19     env = getFreshEnvironment(seed);
20     sim = new Simulation<>(env, maxS, maxT);
21     ...
22   }
23   ...
24   public void performOneStepOfSimulation() {
25     sim.playSingleStepAndWait();
26   }
27   ...
28   public double rval(final int obs) {
29     if (obs >= 0 && obs < StandardProperty.RESERVED_IDS) {
30       switch (StandardProperty.fromInt(obs)) {
31       case TIME:
32         return getTime();
33       case STEP:
34         return sim.getStep();
35       ...
36       default:
37         return getStateEvaluator().getVal(obs, this);
38       }
39     }
40     ...
41   }
42 }
```

Listing 7. AlchemistState extending MULTIVESTA's NewState class

In the constructor (lines 7–11), the superclass initialization is done by a simple `super()` call. The remaining code initializes ALCHEMIST specific parameters such as the maximum time or number of steps to simulate. It is worth noting that those are in general not required, since MULTIVESTA is generally able to detect when the analysis requirements has been met, and consequently stop the simulation flow.

The `setSimulatorForNewSimulation()` method is depitcted in lines 13–22. The goal of the method, invoked by MULTIVESTA before performing a new simulation, is to (re)initialize the status of the simulator, generating a new simulation with the specified seed.

In lines 24–26, `performOneStepOfSimulation()` is provided: resorting to the ALCHEMIST method `playSingleStepAndWait()`, it allows MULTIVESTA to order the execution of a single simulation step.

In order to inspect and analyse the simulation state, the `rval()` method defined in lines 28-41 is invoked. The argument specifies the observation of interest. This method inspects the simulation state for all aspects common to any ALCHEMIST model, e.g. in Listing 7 lines 31–35 are sketched the current simulated time and the number of performed simulation steps. Clearly, each ALCHEMIST model will have its own observations of interest. These are managed resorting to the default branch (lines 36-37), as described in Section IV-B.

```
1  public double getVal(final FourPADProperties prop, final
       ISimulation sim) {
2    final IEnvironment env = sim.getEnvironment();
3    int count = 0;
4    switch (prop) {
5    case PEOPLE_RIGHT:
6      for (final INode<...> node : env) {
7        if (isInRightArea(node, env)) {
8          count++;
9        }
10     }
11     return count;
12   ...
13   case CONNECTIVITY:
14     for (final INode<...> n : env) {
15       count+=env.getNeighborhood(n).getNeighbors().size();
16     }
17     return ((double) count) / env.getNodesNumber();
18   ...
19   default:
20     return 0;
21   }
22 }
```

Listing 8. AlchemistStateEvaluator for the crowd steering model

The resulting integrated tool has been packaged within the standard ALCHEMIST distribution. Simply by downloading ALCHEMIST version 4 or newer, the user is enabled to exploit the analysis capabilities of MULTIVESTA.

### B. Model-specific integration

Depending on the model at hand, it may be necessary to refine the model-independent observations exposed by `AlchemistState` with a set of model-specific ones. This can be done by simply instantiating the `IStateEvaluator` interface provided by MULTIVESTA, constituted by the method `getVal(int observation, NewState state)`.

Listing 8 sketches `AlchemistStateEvaluator`, defined for the crowd steering scenario. For the sake of brevity, only two among all the properties of interest are reported: the number of people that have reached the POI on the right side and the average connectivity of the devices.

## V. ANALYSIS OF THE SCENARIO

This section discusses the analysis performed on our crowd steering scenario, resorting to the integration of MULTIVESTA and ALCHEMIST. The outcome of the analysis is summarized in the three charts of Figure 3, showing, at the varying of the simulated time, the expected values of: the number of people which have reached their POI (top), the average number of connections of the devices (middle), and the number of LSAs in the system (bottom).

The three charts have been obtained by evaluating $MainMQ$ of Listing 9, having 5 parametric temporal operators (lines 1-10). `people@RPOI` and `people@LPOI` regard the number of people which have reached, respectively, the POI on the right and the one on the left. `people@POI` counts instead how many people have reached their destination. The fourth temporal operator (`avgConn`) regards the connectivity degree of the devices. Finally, `LSAs` regards the number of LSA in the system. The temporal operators are analysed at the

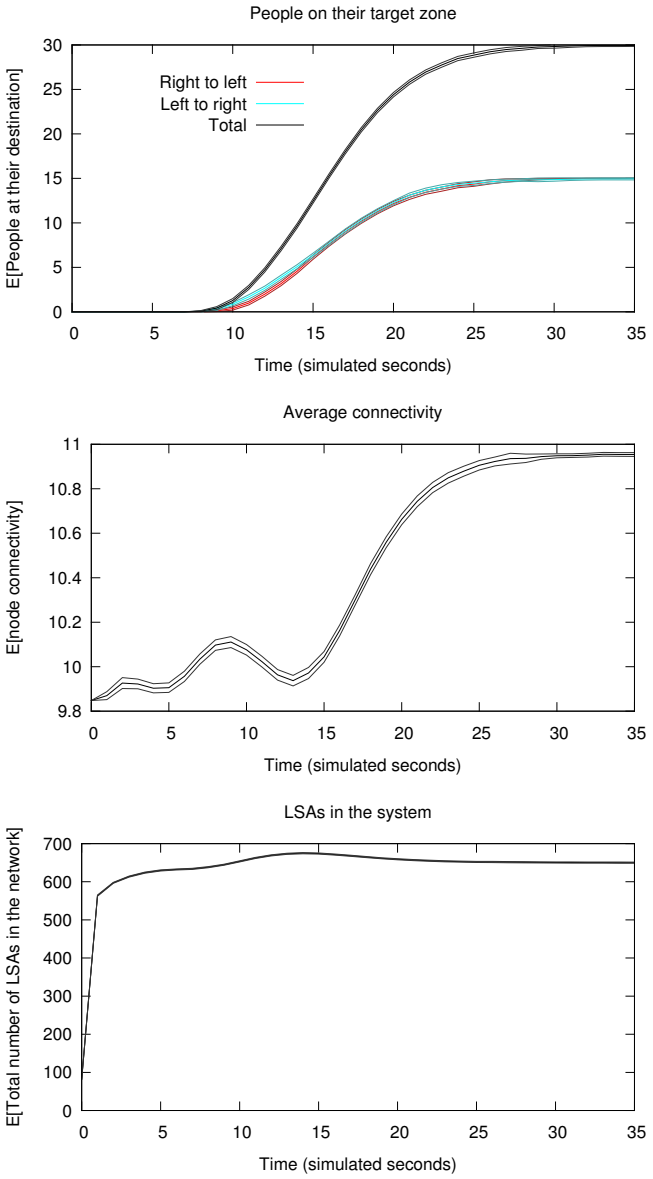


Fig. 3. Analysis of the crowd steering scenario: (top) number of people at the POIs, (middle) average number of connections per device, (bottom) number of LSAs in the system.

varying of the simulated time from 0 to 50 seconds, with step 1 (line 11). Thus the analysis consisted in the estimation of the expected values of the 5 temporal operators instantiated with 50 parameters, for a total of 250 expected values.

A high degree of precision has been required: $\alpha$ has been set to 0.01, while the $\delta$ values (the size of the CIs) have been chosen considering the orders of magnitude of the measures: 0.5 for the instances of `people@POI(`$x$`)`, `people@RPOI(`$x$`)` and `people@LPOI(`$x$`)`; 0.05 for `avgConn(`$x$`)`; and 3 for `LSAs(`$x$`)`. To reach such a level of confidence, the tool ran approximately 2500 simulations, requiring less than a hour.

The discussed confidence intervals are depicted in the aforementioned charts: the two lines drawn above and below the central lines represent the obtained CIs of the expected

```
people@POI(x) = if{s.rval(0) >= x} then s.rval(13)                    1
                                   else #people@POI(x) fi;            2
people@RPOI(x) = if{s.rval(0) >= x} then s.rval(11)                   3
                                    else #people@RPOI(x) fi;          4
people@LPOI(x) = if{s.rval(0) >= x} then s.rval(12)                   5
                                    else #people@LPOI(x) fi;          6
avgConn(x) = if{s.rval(0) >= x} then s.rval(14)                       7
                                 else #avgConn(x) fi;                 8
LSAs(x) = if{s.rval(0) >= x} then s.rval(3)                           9
                              else #LSAs(x) fi;                       10
eval parametric(E[people@POI(x)],E[people@RPOI(x)],                   11
                E[people@LPOI(x)],E[avgConn(x)],E[LSAs(x)],
                x,0.0,1.0,50.0);
```

Listing 9. The evaluated parametric multi-expression ($MainMQ$)

values, thus indicating the intervals in which the actual expected values lie with probability 0.99.

### A. Comments on the obtained results

The top chart regards the first 3 temporal operators: the top plot refers to the number of people in total which have reached their target POI, while the two almost over-imposed lower plots regard the number of people at the right POI and at the left POI. All the three measures under analysis produced monotonically increasing sigmoid curves. We can deduce from this behaviour that there are no systemic errors in the crowd steering system, such as people with no or wrong suggested final destination, in which case we would have noticed one or more flatted zones flawing the sigmoid curve. We note also that after around 30 simulated seconds all the people have reached their target, and that it takes around 10 seconds for the fastest walkers to get to their destination. Note that, despite the analysis has been performed for 50 simulated seconds, the charts presented in Figure 3 have been cut at time 35 to better show the most relevant results: in fact, the system reaches stability after this time, and no event of interest happens later. Moreover we can also notice that people going from left to right are slightly faster than the other group. This difference, due to the asymmetric positioning of people in the centre of the scenario, is hardly visible and would have been impossible to spot with a lower precision analysis.

The middle chart shows the evolution with time of the average number of connections of the devices (i.e. both sensors and smartphones). Since each device is considered to be connected to all those within a range of 1.5 meters, it also gives us a hint about the crowding level. There is a noticeable peak at around 8 seconds: it is due to a high number of people approaching the central group. After that, the peak disappears when most people overtook the obstacle and are walking towards their POI. Finally, there is a growth: progressively, people reach their destination and tend to create a crowd.

The bottom chart shows the number of LSAs in the whole system, indirectly giving hints on the global memory usage. Once the system is started, there is a very quick growth, due to the gradient being spread from the sources to the whole sensors network and to the LSAs produced by the crowd-sensing. The system reaches a substantial stability after a couple of seconds. From that point on, the number of LSAs has very little variations: the system has no "memory leak", in the sense that it does not keep on producing new LSAs without properly removing old data.

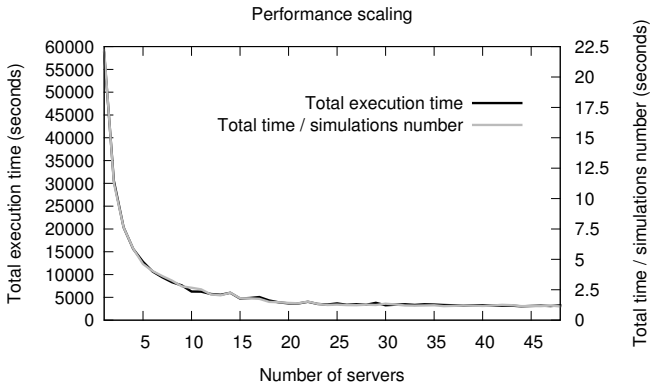


Fig. 4. Performance scaling with of the number of running servers.

### B. Performance assessment

All our experiments have been run on an a machine equipped with four Intel® Xeon® E7540 and 64GB of RAM, running Linux 2.6.32 and Java 1.7.0_04-b20 64bit.

In order to measure the performance scaling of the tool, we ran our analysis multiple times varying the number of MULTIVESTA servers deployed. Results are summarized by the dark line of Figure 4, showing that with only 1 server, the analysis required almost 17 hours, while with more than 30 servers it required less than a hour. For the considered scenario, by distributing simulations we have thus obtained a more than 18 times faster analysis.

It is worth to note that such comparison of performance is affected by the statistical nature of the analysis procedure. Indeed, MULTIVESTA may need to run a sightly different number of simulations in order to obtain the same precision in different evaluations. Intuitively, when evaluating a MULTIQUATEX expression resorting to $x$ or to $x + y$ servers, in both cases we initialize the first $x$ servers with the same $x$ seeds, generating thus the same $x$ simulations. However, the remaining $y$ servers are initialized with new seeds, and thus produce new simulations. Clearly, by having different simulations, we may obtain different sample variances, requiring a different number of simulations. For this reason, the bright line of Figure 4 depicts the time analysis normalized wrt the number of simulations (obtained dividing the overall analysis time by the number of performed simulations). The two lines of Figure 4 evolve accordingly, thus confirming the great performance gain brought by the distribution of simulations.

As discussed in Section III, other than distribution of simulations, MULTIVESTA has a further important feature which dramatically reduces the time necessary to perform analysis tasks: the tool is able to reuse the same simulations to estimate many expected values. The reusage happens on two levels: `(1)` expressions can be made parametric, and thus the same simulations can be used for computing the same property at the varying of a parameter (e.g. at different time steps). An example is $MQ_2$ of Listing 6; `(2)` it is possible to write multi-expressions (e.g. $MQ_{1-2}$ of Listing 4), and thus use the same simulations to obtain the expected values of multiple properties. Moreover, by combining these two features, it is possible to reuse simulations for multiple parametric properties. $MainMQ$ of Listing 9, analysed in this section, is an example of parametric multi-expression. As explicitated by Figure 5 (whose reported analysis have been performed resorting to 48 servers), the parametric expressions (second column) allowed us to save a stunning 96% of execution time wrt the simple expressions case without reuse of simulations (first column). Moreover, the parametric multi-expression feature (third column) allowed us to further cut down this time to a third. We can thus advocate that, by exploiting the feature of reuse of simulations, for the scenario under investigation we obtained a more than 68 times faster analysis.

| | Expressions | Parametric expressions | Parametric multi-expression |
|---|---|---|---|
| `people@POI` | 39999.32 | 2567.26 | n.a. |
| `people@RPOI` | 15891.13 | 1114.86 | n.a. |
| `people@LPOI` | 15560.83 | 950.30 | n.a. |
| `avgConn` | 90111.29 | 1372.64 | n.a. |
| `LSAs` | 58630.45 | 2504.33 | n.a. |
| **Total time** | 220193.02 | 8509.39 | 3215.95 |

Fig. 5. Time performance improvements reusing simulations (seconds).

## VI. CONCLUSIONS AND FUTURE WORKS

This paper presented a novel statistical analysis tool tailored to chemical-inspired pervasive systems. The tool has been obtained by combining ALCHEMIST, an efficient state-of-the-art simulator for chemical-oriented computational systems, with MULTIVESTA, a recently proposed lightweight tool which allows to enrich existing discrete event simulators with efficient distributed statistical analysis capabilities.

By integrating ALCHEMIST with MULTIVESTA, the former has been enriched with capabilities far from those usually offered by simulation tools. Mainly, it is now possible to define in a clean and compact way the properties of interest, to decouple them from the model definition, and to automatize their evaluation. The analysis tasks can be performed efficiently, as the distribution of simulations is supported. Furthermore, another important benefit is the ability of evaluating more properties at once via parametric multi-expressions, thus reducing the number of required simulations and speeding up greatly the analysis operations.

The analysis capabilities and performance of the newly obtained tool have been evaluated in a crowd steering scenario regarding 45 smartphone-equipped people moving to a POI, immersed in an environment with a grid of 200 computationally enabled sensors. The analysis consisted in the estimation, with high degree of precision, of 250 expected values, and required less than an hour in total.

MULTIVESTA already supports several simulators (as discussed in [11]). From now on MULTIVESTA is also part of the ALCHEMIST distribution, and will be used in future for performing analysis on complex scenarios, giving to all the ALCHEMIST users the benefits described in this paper.

## ACKNOWLEDGMENT

**Acknowledgements:** Work supported by the European projects FP7-FET 256873 SAPERE, FP7-FET 257414 ASCENS, and FP7-STReP 600708 QUANTICOL, and by the Italian PRIN 2010LHT4KM CINA.

## REFERENCES


[1] R. Doursat, "Organically grown architectures: Creating decentralized, autonomous systems by embryomorphic engineering," in *Organic computing*. Springer, 2008, pp. 167–199.

[2] M. Viroli and M. Casadei, "Biochemical tuple spaces for self-organising coordination," *Coordination Models and Languages*, pp. 143–162, 2009.

[3] M. Mamei, F. Zambonelli, and L. Leonardi, "Cofields: a physically inspired approach to motion coordination," *Pervasive Computing, IEEE*, vol. 3, no. 2, pp. 52–61, 2004.

[4] F. Zambonelli, G. Castelli, L. Ferrari, M. Mamei, A. Rosi, G. Di Marzo, M. Risoldi, A.-E. Tchao, S. Dobson, G. Stevenson, Y. Ye, E. Nardini, A. Omicini, S. Montagna, M. Viroli, A. Ferscha, S. Maschek, and B. Wally, "Self-aware pervasive service ecosystems," *Procedia Computer Science*, vol. 7, pp. 197–199, Dec. 2011, proceedings of the 2nd European Future Technologies Conference and Exhibition 2011 (FET 11). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050911005667

[5] S. Mariani and A. Omicini, "Molecules of knowledge: Self-organisation in knowledge-intensive environments," *Intelligent Distributed Computing VI*, pp. 17–22, 2013.

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

[7] E. Babulak and M. Wang, "Discrete event simulation: State of the art," *iJOE*, vol. 4, no. 2, pp. 60–63, 2008.

[8] S. Bandini, S. Manzoni, and G. Vizzari, "Agent based modeling and simulation: An informatics perspective," *Journal of Artificial Societies and Social Simulation*, vol. 12, p. 4, 2009. [Online]. Available: http://EconPapers.repec.org/RePEc:jas:jasssj:2009-69-1

[9] C. M. Macal and M. J. North, "Tutorial on agent-based modelling and simulation," *Journal of Simulation*, vol. 4, no. 3, pp. 151–162, 2010.

[10] D. Pianini, S. Montagna, and M. Viroli, "Chemical-oriented simulation of computational systems with Alchemist," *Journal of Simulation*, 2013. [Online]. Available: http://www.palgrave-journals.com/jos/journal/vaop/full/jos201227a.html

[11] S. Sebastio and A. Vandin, "MultiVeStA: Statistical Model Checking for Discrete Event Simulators," submitted to ValueTools 2013.

[12] M. Viroli, D. Pianini, S. Montagna, and G. Stevenson, "Pervasive ecosystems: a coordination model based on semantic chemistry," in *27th Annual ACM Symposium on Applied Computing (SAC 2012)*, S. Ossowski, P. Lecca, C.-C. Hung, and J. Hong, Eds. Riva del Garda, TN, Italy: ACM, 26–30 Mar. 2012. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2245276.2245336

[13] M. Schumacher, L. Grangier, and R. Jurca, "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. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 163–172. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-75254-7_17

[14] S. Bandini, S. Manzoni, and G. Vizzari, "Crowd Behavior Modeling: From Cellular Automata to Multi-Agent Systems," in *Multi-Agent Systems: Simulation and Applications*, ser. Computational Analysis, Synthesis, and Design of Dynamic Systems, A. M. Uhrmacher and D. Weyns, Eds. CRC Press, Jun. 2009, ch. 13, pp. 389–418. [Online]. Available: http://crcpress.com/product/isbn/9781420070231

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

[16] E. Sklar, "Netlogo, a multi-agent simulation environment," *Artificial Life*, vol. 13, no. 3, pp. 303–311, 2007.

[17] C. Priami, "Stochastic pi-calculus," *Comput. J.*, vol. 38, no. 7, pp. 578–589, 1995.

[18] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989. [Online]. Available: http://dx.doi.org/10.1109/5.24143

[19] A. M. Uhrmacher and C. Priami, "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, 2005, pp. 317–326. [Online]. Available: http://dl.acm.org/citation.cfm?id=1162708.1162767

[20] R. Ewald, C. Maus, A. Rolfs, and A. M. Uhrmacher, "Discrete event modeling and simulation in systems biology," *Journal of Simulation*, vol. 1, no. 2, pp. 81–96, 2007. [Online]. Available: http://dx.doi.org/10.1057/palgrave.jos.4250018

[21] D. T. Gillespie, "Exact stochastic simulation of coupled chemical reactions," *The Journal of Physical Chemistry*, vol. 81, no. 25, pp. 2340–2361, December 1977. [Online]. Available: http://dx.doi.org/10.1021/j100540a008

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

[23] S. Montagna, D. Pianini, and M. Viroli, "A model for drosophila melanogaster development from a single cell to stripe pattern formation," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. ACM, 2012, pp. 1406–1412.

[24] S. Bandini, S. Manzoni, and G. Vizzari, "Crowd modeling and simulation: Towards 3d visualization," in *Recent Advances in Design and Decision Support Systems in Architecture and Urban Planning*, J. P. Leeuwen and H. J. Timmermans, Eds. Springer Netherlands, 2005, pp. 161–175. [Online]. Available: http://link.springer.com/chapter/10.1007/1-4020-2409-6_11

[25] J. Beal, J. Bachrach, D. Vickery, and M. Tobenkin, "Fast self-healing gradients," in *Proceedings of the 2008 ACM symposium on Applied computing*. ACM, 2008, pp. 1969–1975.

[26] J. Beal, "Flexible self-healing gradients," in *Proceedings of the 2009 ACM symposium on Applied Computing*. ACM, 2009, pp. 1197–1201.

[27] J. L. Fernandez-Marquez, G. D. M. Serugendo, S. Montagna, M. Viroli, and J. L. Arcos, "Description and composition of bio-inspired design patterns: a complete overview," *Natural Computing*, pp. 1–25, 2013.

[28] S. Montagna, M. Viroli, M. Risoldi, D. Pianini, and G. Di Marzo Serugendo, "Self-organising pervasive ecosystems: a crowd evacuation example," *Software Engineering for Resilient Systems*, pp. 115–129, 2011.

[29] S. Montagna, D. Pianini, and M. Viroli, "Gradient-based self-organisation patterns of anticipative adaptation," in *Self-Adaptive and Self-Organizing Systems (SASO), 2012 IEEE Sixth International Conference on*. IEEE, 2012, pp. 169–174.

[30] M. Chraibi, M. Freialdenhoven, A. Schadschneider, and A. Seyfried, "Modeling the desired direction in a force-based model for pedestrian dynamics," *arXiv preprint arXiv:1207.1189*, 2012.

[31] K. Sen, M. Viswanathan, and G. Agha, "Vesta: A statistical model-checker and analyzer for probabilistic systems," in *Quantitative Evaluation of Systems, 2005. Second International Conference on the*, 2005, pp. 251–252.

[32] M. AlTurki and J. Meseguer, "Pvesta: A parallel statistical model checking and quantitative analysis tool," in *CALCO*, ser. Lecture Notes in Computer Science, A. Corradini, B. Klin, and C. Cîrstea, Eds., vol. 6859. Springer, 2011, pp. 386–392.

[33] G. A. Agha, J. Meseguer, and K. Sen, "PMaude: Rewrite-based specification language for probabilistic object systems," in *QAPL 2005*, ser. ENTCS, A. Cerone and H. Wiklicky, Eds., vol. 153(2). Elsevier, 2006, pp. 213–239.

[34] H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability," *Formal Asp. Comput.*, vol. 6, no. 5, pp. 512–535, 1994.

[35] A. Aziz, V. Singhal, and F. Balarin, "It usually works: The temporal logic of stochastic systems," in *CAV*, ser. Lecture Notes in Computer Science, P. Wolper, Ed., vol. 939. Springer, 1995, pp. 155–165.

[36] C. Baier, J.-P. Katoen, and H. Hermanns, "Approximate symbolic model checking of continuous-time markov chains," in *CONCUR*, ser. Lecture Notes in Computer Science, J. C. M. Baeten and S. Mauw, Eds., vol. 1664. Springer, 1999, pp. 146–161.

[37] K. Sen, M. Viswanathan, and G. Agha, "On statistical model checking of stochastic systems," in *CAV*, ser. Lecture Notes in Computer Science, K. Etessami and S. K. Rajamani, Eds., vol. 3576. Springer, 2005, pp. 266–280.