A Self-Organising Infrastructure for Chemical-Semantic Coordination: Experiments in TuCSoN

Elena Nardini Mirko Viroli Matteo Casadei Andrea Omicini
ALMA MATER STUDIORUM – Università di Bologna
(elena.nardini, mirko.viroli, m.casadei, andrea.omicini)@unibo.it

***Abstract*—Recent works proposed the adoption of a nature-inspired approach of chemistry for implementing service architectures suitable for pervasive applications [34]. In particular, [31] proposes a chemical-semantic tuple-space model where coordination of data, devices and software agents – representing the services of the pervasive computing application – are reified into proper tuples managed by the coordination infrastructure. Service coordination is enacted by chemical-like reactions that semantically match those tuples accordingly enacting the desired interaction patterns (composition, aggregation, competition, contextualisation, diffusion and decay).**

After showing and motivating the proposed coordination approach for situated, adaptive, and diversity-accomodating pervasive computing systems, in this paper we outline how it is possible to concretise the approach on the TuCSoN coordination infrastructure, which can been suitably enhanced with modules supporting fuzzy semantic-coordination and execution engine for chemical-inspired coordination laws.

I. INTRODUCTION

The advent of ubiquitous wireless connectivity and computing technologies like pervasive services and social networks will re-shape the Information and Communication Technology (ICT) landscape. In particular, new devices with increasing interaction capabilities will be exploited to create services able to inject and retrieve data from any location of the very dynamic and dense network that will pervade our everyday environments. Such a scenario calls for infrastructures promoting a concept of pervasive "eternality": changes in topology, device technology, and continuous injection of new services have to be dynamically incorporated with no significant re-engineering costs at the middleware level [35], [34]. In this context, self-organisation – supporting situatedness, adaptivity and long-term accommodation of diversity – will increasingly be required as a key system-property for the system-component coordination.

Among the available metaphors – e.g. physical, chemical, biological, social [34] – used as a source of inspiration for self-organising computational mechanisms, here we focus on chemistry. In particular, we adopt the concept of chemical-semantic tuple-spaces introduced in [31] as a coordination model, extending the standard tuple space model that is frequently adopted in middleware for situated and adaptive pervasive computing [28], [12], [19], [20]. In chemical-semantic tuple-spaces, tuples are seen as sort of species in a population, and coordination laws taking the form of chemical reactions semantically apply to such tuples and evolve them over time. Such an evolution behaviour promotes the exploitation of (natural or idealised) chemical reactions that are known to make interesting self-organisation properties emerge [31].

This paper focusses on how a distributed architecture for chemical-semantic tuple-spaces can be implemented, namely, in terms of a coordination infrastructure providing the *fabric* of chemical tuples along with the stochastic and semantic application of chemical-like laws. As a basis for this implementation we start from the TuCSoN coordination infrastructure [24] supporting the notion of *semantic tuple-centre* [21], i.e. programmable semantic tuple-space. In particular, TuCSoN promotes a view of the tuple space as a set of facts in a logic theory, and its program as a set of rules dictating how the tuple set should evolve as time passes and as new interaction events occur. TuCSoN appears a suitable means to enact the two basic ingredients necessary to implement chemical tuple-spaces in TuCSoN: *(i)* chemical-inspired stochastic evolution of tuples, which is achieved by implementing the well-known Gillespie's exact simulation algorithm [16] as a tuple space program, and *(ii)* fuzzy semantic-matching of chemical laws against tuples in the space obtained seeing a tuple as an individual of an ontology, and a reactant in the chemical reaction as a concept of the ontology [17].

The remainder of this paper is organised as follows. Section II motivates the proposed approach. Section III introduces the chemical-semantic coordination model and provides abstract examples of coordination laws. Section IV describes how the TuCSoN coordination infrastructure can be tailored to support the proposed model. Finally, Section V outlines related works in coordination and middleware for self-adaptive and self-organising systems, and Section VI concludes discussing a roadmap towards completing the development of the infrastructure.

II. CHEMICAL-SEMANTIC TUPLE-SPACES FOR PERVASIVE SERVICES

In order to better explain the motivation behind the model presented in this paper, we rely on a case study, which we believe well represents a large class of pervasive computing applications in the near future. We consider a pervasive display

infrastructure, used to surround our environments with digital displays, from those in our wearable devices and domestic hardware, to wide wall-mounted screens that already pervade urban and working environments [13]. In particular, as a reference domain, we consider an airport terminal filled with wide screens mounted in the terminal area, i.e in the shops, in the corridors and in the gates, down to tiny screens installed in each seat of the gate areas or directly on passengers' PDAs.

Situatedness. Information should be generally displayed based on the current state of the surrounding physical and social environment like surrounding temperature sensors or passenger profiles/preferences. Hence, services should be able to interact with the surrounding physical and social world, accordingly adapting their behaviour.

Adaptivity. Complementary to the above, the display infrastructure, and the services within it, should be able to automatically adapt to changes and contingencies in an automatic way. For instance, when a great deal of new information to be possibly displayed emerges, the displayed information should overall spontaneously re-distribute and re-shape across the set of existing local displays.

Diversity. The service infrastructure should not limit the number and classes of services potentially provided, but rather taking advantage of the injection of new services by exploiting them to improve and integrate existing services whenever possible. For example, the display infrastructure also should be able to allow users – other than display owners – to upload information, like user private-content uploaded from her/his PDA, to displays so as to enrich the information offer or adapt it to their own needs.

As it is shown in many proposals for pervasive computing environments and middleware infrastructures, situatedness and adaptiveness are promoted by the adoption of shared virtual spaces for services and component interaction [28], [12], [19], [20], [31]. Among the various shared-virtual-space models proposed in literature, we adopt the chemical tuple-space model [31], in which tuples – containing semantic information about the "entities" to be coordinated (services, devices, data) – evolve in a stochastic and spatial way through coordination laws resembling chemical reactions. We observe that this model can properly tackle the requirements sought for adaptive pervasive services.

Concerning *situatedness*, the current situation in a system locality is represented by the tuples existing in the tuple space. Some of them can act as catalysts for specific chemical reactions, thus making system evolution intrinsically context-dependent. Concerning *adaptivity*, it is known from biology that some complex chemical systems are auto-catalytic (i.e. they produce their own catalyst), providing positive-negative feedbacks that induce self-organisation [11] and lead to the spontaneous toleration of environment perturbations. Such systems can be modelled by simple idealised chemical reactions – e.g. prey-predator systems, Brussellator, and Oregonator [16] – regarded as a set of coordination laws for pervasive services. Finally, considering *diversity*, we note that chemical reactions follow a simple pattern: they have some reactants (typically 1 or 2) which combine, resulting in a set of products, through a propensity (or rate) dictating the likelihood for this combination to actually happen. Similarly to the natural chemistry, in our framework this generates chemical reactions that can be instantiated for the specific and unforeseen services that will be injected in the system over time, using semantic matching.

Though key requirements seem to be supported in principle, designing the proper set of chemical reactions to regulate system behaviour is crucial. Without excluding the appropriateness of other solutions, in this paper we mostly rely on chemical reactions resembling laws of population dynamics as, e.g. the prey-predator system [16], [7]. This kind of idealised chemical reactions has been successfully used to model auto-catalytic systems manifesting self-organisation properties: but moreover, they can also nicely fit the "ecological" metaphor that is often envisioned for pervasive computing systems [5], [30], [1], [34], [35]—namely, seeing pervasive services as spatially situated entities living in an ecosystem of other services and devices.

III. The Coordination Model of Chemical-Semantic Tuple-Spaces

In this section, first we informally introduce the coordination model of chemical-semantic tuple-spaces (Section III-A), then describe some example applications in the context of competitive pervasive services (Section III-B).

A. Coordination Model

The chemical-semantic tuple-space model is an extension of standard LINDA settings with multiple tuple spaces [15]. A LINDA tuple space is simply described as a repository of tuples (structured data chunks like records) for the coordination of external "agents", providing primitives used respectively to insert, read, and remove a tuple. Tuples are retrieved by specifying a tuple template—a tuple with wildcards in place of some of its arguments. The proposed model enhances this basic schema with the following ingredients.

Tuple concentration and chemical reactions. We attach an integer value called "concentration" to each tuple, measuring the pertinence/activity value of the tuple in the given tuple space: the higher such a concentration, the more likely and frequently the tuple will be retrieved and selected by the coordination laws to influence system behaviour. Tuple concentration 'spontaneously" evolves, as typically the pertinence of system activities is. This is achieved by coordination rules in the form of chemical reactions—the only difference with respect to standard chemical reactions is that they now specify tuple templates instead of molecules. For example, a reaction "$X + Y \xrightarrow{0.1} X + X$" would mean

that tuples x and y matching X and Y are to be selected, get combined, and as a result one concentration item of y turns into x—concentration of y decreases by one, concentration of x increases by one. According to [16], this transition is modelled as a Poisson event with average rate (i.e. frequency) $0.1 \times \#x \times \#y$ ($\#x$ is the concentration of x). This model makes a tuple space running as a sort of exact chemical simulator, picking reactions probabilistically: external agents observing the evolution of tuples would perceive something equivalent to the corresponding natural/artificial chemical system described by those reactions.

Semantic matching. It is easy to observe that standard syntactic matching for tuple spaces can hardly deal with the openness requirement of pervasive services, in the same way as syntactic match-making has been criticised for Web services [25]. This is because we want to express general reactions that apply to specific tuples independently of their syntactic structure, which cannot be clearly foreseen at design time. Accordingly, *semantic matching* can be considered as a proper matching criterion for our coordination infrastructure [25], [4], [9].

It should be noted that matching details are orthogonal to our model, since the application at hand may require a specific implementation of them—e.g. it strongly depends on the description of application domain. As far as the model is concerned, we only assume that matching is fuzzy [9], i.e. matching a tuple with a template returns a "vagueness" value between 0 and 1, called *match degree*. Vagueness affects the actual application rate of chemical reactions: given a chemical reaction with rate r, and assume reactants match some tuples with degree 0.5, then the reaction can be applied to those tuples with an actual rate of $0.5 * r$, implying a lower match likelyhood—since match is not perfect. Namely, the role of semantic matching in our framework is to allow for coding general chemical laws that can uniformly apply to specific cases—appropriateness influences probability of selection.

Tuple transfer. We add a mechanism by which a (unit of concentration of a) tuple can be allowed to move towards the tuple space of a neighbouring node, thus generating a *computational field*, namely, a data structure distributed through the whole network of tuple spaces. Accordingly, we introduce the notion of "firing" tuple (denoted $t^{\rightsquigarrow}$), which is a tuple (produced by a reaction) scheduled for being sent to a neighbouring tuple space—any will be selected non-deterministically. For instance, the simple reaction "$X \xrightarrow{0.1} X^{\rightsquigarrow}$" is used to transfer items of concentration of any tuple matching X out from the current tuple space.

B. Examples

We now discuss some examples of chemical reactions enacting general coordination patterns of interest for pervasive service systems.

Local competition. We initially consider a scenario in which a single tuple space mediates the interactions between pervasive services and their users in an open and dynamic system. We aim at enacting the following behaviour: *(i)* services that do not attract users fade until eventually disappearing from the system, *(ii)* successful services attract new users more and more, and accordingly, *(iii)* overlapping services compete one another for survival, so that some/most of them eventually come to extinction.

An example protocol for service providers can be as follows. A tuple `service` is first inserted in the space to model publication, specifying service identifier and semantic description of the service content. Dually, a client inserts a specific request as a tuple `request`—insertion is the (possibly implicit) act of publishing user preferences. The tuple space is charged with the role of matching a request with a reply, creating a tuple `toserve(service,request)`, combining a request and a reply semantically matching. Such tuples are read by the service provider, which collects information about the request, serves it, and eventually produces a result emitted in the space with a tuple `reply`, which will be retrieved by the client. The abstract rules we use to enact the described behaviour are as follows:

$$\texttt{(USE)} \quad \texttt{SERV} + \texttt{REQ} \xmapsto{u} \texttt{SERV} + \texttt{SERV} + \texttt{toserve(SERV,REQ)}$$

$$\texttt{(DECAY)} \quad \texttt{SERV} \xmapsto{d} 0$$

On the left side (reactants), `SERV` is a template meant to match any service tuple, `REQ` a template matching any request tuple; on the right side (products), `toserve(SERV,REQ)` will create a tuple having in the two arguments the service and request tuples selected, while `0` means there will be no product. Rule (USE) has a twofold role: *(i)* it first selects a service and a request, it semantically matches them and accordingly creates a `toserve` tuple, and dynamically removes the request; and *(ii)* it increases service concentration, so as to provide a positive feedback—resembling the prey-predator system described by Lotka-Volterra equations [7], [16]. We refer to *use rate* of a couple service/request as u multiplied by the match degree of those reactants when applying (USE) law, as described in the previous section: as a result, it can be noted that the higher the match degree, the more likely a service and a request are combined. On the other hand, rule (DECAY) makes any concentration item of the service tuple disappear at rate d, contrasting the positive feedback of (USE): here, the overall *decay rate* of a service is d multiplied by the match degree—with no match, we would have no decay at all.

Spatial competition. This example can be extended to a network of tuple spaces, so as to emphasise the spatial and context-dependent character of competing services. Suppose each space is programmed with (USE,DECAY) reactions plus a simple diffusion law for service tuples:

$$\texttt{(DIFFUSE)} \quad \texttt{SERV} \xmapsto{m} \texttt{SERV}^{\rightsquigarrow}$$

The resulting system can be used to coordinate a pervasive service scenario in which a service is injected into a node of the network (e.g. the node where service is more urgently needed, or where the producer resides), and accordingly starts diffusing around on a step-by-step basis until possibly covering the whole network—hence becoming a global service. This situation is typical in the pervasive display infrastructure, since a frequent policy for visualisation services would be to show them on any display of the network—although more specific policies might be enacted to make certain services only locally diffuse.

In this system, we can observe the dynamics by which the injection of a new and improved service may eventually result in a complete replacement of previous versions—spatially speaking, the region where the new service is active is expected to enlarge until covering the whole system, while the old service diminishes. In the context of visualisation services, for instance, this would amount to the situation where an existing advertisement service is established, but a new one targeted to the same users is injected that happens to have greater use rate, namely, it is more appropriate for the average profile of users: we might expect this new service to overcome the old one, which accordingly extinguishes.

IV. An architecture based on the TuCSoN infrastructure

In this section we show that an infrastructure for the chemical tuple space model can be implemented on top of an existing tuple space middleware, such as TuCSoN. In particular, after a general overview of TuCSoN (Section IV-A), we describe how the two basic additional ingredients of the proposed model can be supported on top of TuCSoN: semantic matching (Section IV-B) and chemical engine (Section IV-C).

A. Overview of TuCSoN

TuCSoN (Tuple Centres Spread over the Network) [24] is a coordination infrastructure that manages the interaction space of an agent-based system by means of ReSpecT [23] *tuple centres*, which are *Linda* tuple spaces [15] empowered with the ability to define their behaviour in response / reaction to communication events. Other than supporting the notion of tuple centre, TuCSoN also extends Linda model from the topology viewpoint. Differently from the [15] tuple space model, TuCSoN has been conceived for providing a suitable infrastructural support for engineering distributed application scenarios [24]. In particular, tuple centres are distributed through the network, hosted in the nodes of the infrastructure, and organised into articulated domains, each characterised by a gateway node and a set of nodes called places. A place is meant to host tuple centres for the specific applications/systems, while the gateway node is meant to host tuple centres used for domain administration, keeping information on the places.

As discussed in [33], TuCSoN supports features that are key for implementing self-organising systems, some of which are here recapped that are useful for implementing the chemical tuple space model:

Topology and locality. Tuple centres can be created locally to a specific node, and the gateway tuple centre can be programmed to keep track of which tuple centres reside in the neighbourhood—accessible either by agents or by tuple centres in current node.

On-line character and time. TuCSoN supports the so called "on-line coordination services", executed by reactions that are fired in the background of normal agent interactions, through timed reactions.

Probability. Probability is a key feature of self-organisation, which is necessary to abstractly deal with the unpredictability of contingencies in pervasive computing. In TuCSoN this is supported by drawing random numbers and using them to drive the reaction firing process, that is, making tuple transformation be intrinsically probabilistic.

B. Semantic Matching

We now describe the extension of the tuple centre model that allows us to perform semantic reasoning over tuples, namely, the ability of matching a tuple with respect to a template not only syntactically as usual, but also semantically. In spite some approaches have been proposed to add semantic reasoning to tuple spaces [22], we here introduce a different model, which is aimed at smoothly extending the standard settings of tuple spaces [21].

Abstract model. From an abstract viewpoint, a tuple centre can be seen as a knowledge repository structured as a set of tuples. According to a semantic view, such knowledge represents a set of objects occurring in the application domain, whose meaning is described by an ontology, that is, in terms of concepts and relations among them.

In order to formally define the notions of domain ontology and objects, the literature makes available a family of knowledge representation formalisms called *Description Logics* (DL) [2]—we rely on *SHOIN(D)* Description Logic, which represents the theoretical counterpart of W3C's standard *OWL* DL [17]. In DL, an ontology component called *TBox* is first introduced that includes the so-called terminological axioms: *concept* descriptions (denoting meaningful sets of individuals of the domain), and *role* descriptions (denoting relationships among individuals). Concepts can be of the following kinds: $\top$ is the set of all objects, $\bot$ the void set, $C \sqcup D$ is union of concepts, $C \sqcap D$ intersection, $\neg D$ negation, $\{i_1, \ldots, i_n\}$ is a set of individuals, $\forall R.C$ is the set of objects that are in relation (through role R) with only objects belonging to concept C, $\exists R.C$ is the set of objects that are in relation (through role R) with at least one object belonging to concept C, and $\leq nR$ is the set of objects that are in relation (through role R) with no more than n objects (and similarly for concepts $\geq nR$ and $= nR$). Given these constructs, the TBox provides axioms

for expressing inclusion of concepts ($C \sqsubseteq D$). For instance, a TBox for a *car domain* [9] can provide the following assertions

$$\begin{aligned} \mathit{MaxSpeed} &\sqsubseteq \{\texttt{90km/h}, \texttt{180km/h}, \texttt{220km/h}, \\ &\quad \texttt{280km/h}\} \\ \mathit{Car} &\sqsubseteq (=1\mathit{hasMaxSpeed}) \\ (=1\mathit{hasMaxSpeed}) &\sqsubseteq \mathit{Car} \\ \top &\sqsubseteq \exists\, \mathit{hasMaxSpeed}.\mathit{MaxSpeed} \\ \mathit{SlowCar} &\sqsubseteq \mathit{Car} \sqcap (\exists\, \mathit{hasMaxSpeed}.\{\texttt{90km/h}\}) \\ \mathit{CityCar} &\sqsubseteq \mathit{SlowCar} \end{aligned}$$

which respectly: *(i)* defines the concept *MaxSpeed* as including 4 individuals, *(ii)* defines the concept *Car* and states that all its objects have precisely one maximum speed value, *(iii)* conversely states that any object with one maximum speed value is a car, *(iv)* states that maximum speed value is an object of *MaxSpeed*, *(v)* defines *SlowCar* as a new concept (sub-concept of *Car*) such that its individuals have `90km/h` as maximum speed, and finally *(vi)* defines a *CityCar* as a kind of slow car—possibly to be completed with other features, e.g. being slow, compact and possibly featuring electric battery. By these kinds of inclusion, which are typical of the OWL standard approach, one can flexibly provide suitable definitions for concepts of sport cars, city cars, and so on.

Another component of DL is the so-called *ABox*, defining axioms to assert specify domain objects and their properties: they can be of kind $C(a)$, declaring individual a and the concept C it belongs to, and of kind $R(a, b)$, declaring that role R relates individual a with b. Considering the car domain example, the ABox could include axioms $\mathit{Car}(\texttt{f40})$, $\mathit{Car}(\texttt{fiat500})$, $\mathit{hasMaxSpeed}(\texttt{f40}, \texttt{280km/h})$, and $\mathit{hasMaxSpeed}(\texttt{fiat500}, \texttt{90km/h})$.

Semantic reasoning of DL basically amounts – among the others – to check whether an individual belongs to a concept, namely, the so-called semantic matching. As a simple example, a DL checker could verify that `fiat500` is an instance of *CityCar*. This contrasts syntactic matching, which would have failed since *Car* and *CityCar* are two "types" that do not syntactically match—they rather semantically match due to the definitions in the TBox.

Given the above concepts of DL, we design the semantic extension of tuple centres by the following ingredients, which will be described in more detail in turn: *(ontologies)* an ontology has to be attached to a tuple centre, so as to ground the definition of concepts required to perform semantic reasoning; *(semantic tuples)* a semantic tuple represents an individual, and a language is hence to be introduced to specify individual's name, the concept it belongs to, and the individuals it is related to by roles; *(tuple templates)* templates are to be used to flexibly retrieve tuples, hence we link the semantic tuple template notion with that of concept in the ontology, and accordingly introduce a template language; *(matching mechanism)* matching simply amounts to check whether the tuple is an instance of the concept described by the template, providing a match factor in between 0 and 1. In order to give a "fuzzy" notion of matching we take inspiration from the work shown in [9] – describing a fuzzy extension of DL – introducing the notion of *degree* (a number in between 0 and 1) into the language of tuples and templates.

Ontology language. In our implementation we adopt the OWL [17] ontology language in order to define domain ontologies in TuCSoN. OWL is an XML-based ontology language introduced by W3C for the Semantic Web: relying on a standard language for ontologies is key for the openness aims of the application domains considered in this paper, and moreover, standard automated reasoning techniques can be exploited relying on existing open source tools—like e.g. *Pellet* reasoner [29]. Accordingly, each tuple centre carries an OWL ontology describing the TBox, and that internal machinery can be easily implemented so as to query and semantically reason about it.

Semantic tuple language. Instead of completely departing from the syntactic setting of TuCSoN, where tuples are expressed as first-order terms, we design a smooth extension of it, so as to capture a rather large set of situations. The car domain example described above would be expressed by the semantic tuple `fiat500:'Car'(hasMaxSpeed:'90km/h')`. Namely, the tuple describes individual name, concept name, and list of role fillers—extending the case of first-order tuples, where each tuple is basically the specification of a "type" (functor name) followed by an ordered list of parameters. By exploiting the definition of *degree* introduced in [9], we can also write:

```
f40:'SportCar'->0.8(
              hasMaker : ferrari->0.8,
              hasMaxSpeed : '285km/h',
              hasColour in {red->0.3,
                            black->0.7})
```

where "`->N`" represents the *degree* of belonging of an individual to a concept or to a relationship with other individuals. Where a "`->N`" is not defined, the *degree* is 1. Note that semantic tuples are basically first-order terms with the introduction of few infix binary functors ("`:`", "`in`" and "`->`").

Semantic templates language. Another aspect to be faced concerns the representation of semantic tuple templates as specifications of sets of domain individuals (concepts) among which a matching tuple is to be retrieved. The grammar of tuple templates we adopt basically turns DL concepts into a term-like syntax (as in Prolog), extended with the possibility to associate to each concept a *degree* representing how much individuals belong to a particular concept and to describe a

weighted sum of concepts, as shown in [9]:

$$
\begin{array}{rcl}
C & ::= & \texttt{'\$ALL'} \mid \texttt{'\$NONE'} \mid \texttt{cname} \mid C\texttt{,}C \mid C\texttt{;}C \mid \\
 & & \texttt{not(}C\texttt{)} \mid \{\texttt{inamelist}\} \mid CR \mid C\texttt{->}N \mid \\
 & & C\texttt{->}N1 + ... + C\texttt{->}Nn \\
 & & (where N1 + ... + Nn = 1) \\
 & & \\
CR & ::= & [\texttt{exists}|\texttt{only}](\texttt{pname in } C) \mid \\
 & & \#\gtrless N : \texttt{pname}
\end{array}
$$

Elements `cname`, `iname`, and `pname` are constants terms, expressing names of concepts, individuals and properties. Following the grammar, concepts orderly express *all* individuals, *no* individual, a concept name, intersection, union, negation, an individual list, or a concept specified via role-fillers. Examples of the latter include: "`exists` P `in` C" (meaning $\exists\, P.C$), "`only` P `in` C", (meaning $\forall\, P.C$), "P `in` C"(meaning $\exists\, P.C \sqcap \forall\, P.C$), "$\# \geq$ `2:P`" (meaning $\geq 2P$). Additional syntactic sugar is used: "`exists` P`:`i" stands for "`exists` P `in` $\{i\}$", and "C`(`CR_1`,..,`CR_n`)`" stands for "C`,`CR_1`,..,`CR_n". Examples of semantic templates are as follows:

```
'Car'->0.9(exists hasMaxSpeed:
                  {'90km/h','280km/h'})

'Car'(#>1:hasEnergyPower),
        ((hasMaker:ford)->0.9;
        (hasMaker:ferrari)->0.6)
```

The former specifies those individuals that are cars with a degree 0.9, having either `90km/h` or `280km/h` maximum speed, the latter those cars that come with at least two choices of energy power and that have either `ford` or `ferrari` maker, respectively with degree 0.9 and 0.6.

Semantic matching. In order to enable semantic support in TuCSoN, a tuple centre has to be related to an ontology, to which semantic tuples refer to. In order to encapsulate an ontology, tuple centres exploit the aforementioned *Pellet* reasoner [29]—an open-source DL reasoner based on OWL and written in Java likewise TuCSoN. In particular, *Pellet* can load an OWL TBox and an ABox, and provides the *Jena-API* in order to add and remove individuals by its own ABox.

Hence, each semantic tuple is carried not only in the tuple space likewise syntactic tuples, but also in the ontology ABox, after it has been *defuzzificated*, in order to support reasoning. The reasoner is internally called each time we are checking for a semantic match: the semantic template is decomposed and converted into a set of SPARQL queries (the language used by Jena-API). Each query corresponds to a concept defined in the semantic template. The results of the set of SPARQL queries is combined with the operators used in the semantic template and for each individual obtained by the combination a degree (a number in between 0 and 1) is calculated by exploiting the fuzzy operators defined in [9]. Then, a set of individuals with a degree associated is retrieved. This behaviour is embedded in the tuple centre, such that each time a semantic template is specified into a retrieval operation, *any* semantic tuple can actually be returned—namely, the standard behaviour is still non-deterministic. With the classical operations *in* and *rd*, the individual with maximum degree is retrieved, but by exploiting the programmability of tuple centres, it is also possible to retrieve a ranked list of the individuals that satisfy a semantic template.

C. Chemical Reactions

We now describe how a TuCSoN tuple centre can be specialised to act as a chemical-like system where semantic tuples play the role of reactants, which combine and transform over time as occurring in chemistry.

Coding reactants and laws. Tuples modelling reactant individuals are kept in the tuple space in the form `reactant(X,N)`, where `X` is a semantic tuple representing the reactant and `N` is a natural number denoting concentration. Laws modelling chemical reactions are expressed by tuples of the form `law(InputList,Rate,OutputList)`, where `InputList` denotes the list of the reacting individuals, and `Rate` is a float value specifying the constant rate of the reaction. As a reference example, consider the chemical laws resembling (USE,DECAY) rules in previous section: $S + R \xrightarrow{10.0} S + S$ and $S \xrightarrow{10} 0$, where S and R represent semantic templates for services and requests. Such laws can be expressed in TuCSoN by tuples `law([S,R],10,[S,S])` and `law([S],10,[])`. On the other hand, tuples `reactant(sa,1000)`, `reactant(sb,1000)` and `reactant(r,1000)` represent reactants for two semantic tuples (`sa` and `sb`) matching `S`, and one (`ra`) matching `R`. The set of enabled laws at a given time is conceptually obtained by instantiating the semantic templates in reactions with all the available semantic tuples. In the above case they would be `law([sa,r],r1a,[sa,sa])`, `law([sb,r],r1b,[sb,sb])`, `law([sa],r2a,[])` and `law([sb],r2b,[])`. The rate of each enabled reaction (usually referred to as *global rate*) is obtained as the product of chemical rate and match degree as described in Section III. For instance, rate `r1a` can be calculated as $10.0 \times \#\texttt{sa} \times \#\texttt{r} \times \mu(\texttt{S} + \texttt{R}, \texttt{sa} + \texttt{r})$, where μ is the function returning the match factor between the list of semantic reactants and the list of actual tuples.

As an additional kind of chemical law, it is also possible to specify a transfer of molecules towards other tuple centres by a law of the kind `law([X],10,[firing(X)])`.

ReSpecT engine. The actual chemical engine is defined in terms of ReSpecT reactions, which can be classified according to the provided functionality. As such, there are reactions for *(i)* managing chemical laws and reactants, i.e. ruling the dynamic insertion/removal of reactants and laws, *(ii)* controlling engine start and stop, *(iii)* choosing the next chemical law to be executed, and *(iv)* executing chemical laws. For the sake of conciseness we only describe part (iii), which

is the one that focusses on Gillespie's algorithm for chemical simulations [16].

This computes the choice of the next chemical law to be executed, based on the following ReSpecT reaction, triggered by operation `out(engine_trigger)` which starts the engine.

```
reaction( out(engine_trigger), endo, (
          in(engine_trigger),
          chooseLaw(law(IL,_,OL),Rtot),
          rand_float(Tau),
          Dt is round((log(1.0/Tau)/Rtot)*1000),
          event_time(Time), Time2 is Time + Dt,
          out_s(reaction( time(Time2),
          endo, out(engine_trigger) )),
          out(execution(law(IL,_,OL),Time))
)).
```

First of all, a new law is chosen by the `chooseLaw` predicate, which returns `Rtot`, the global rate of all the enabled chemical laws, and a term `law(IL _,OL)`—`IL` and `OL` are bound respectively to the list of reactants and products in the chosen law, after templates are instantiated to tuples as described above. Then, according to Gillespie algorithm, time interval `Dt` – denoting the overall duration of the chemical reaction – is stochastically calculated (in milliseconds) as $log(1/Tau)/Rtot$, where Tau is a value randomly chosen between 0 and 1 [16]. A new timed reaction is accordingly added to the ReSpecT specification and will be scheduled for execution `Dt` milliseconds later with respect to `Time`, which is the time at which `out(engine_trigger)` occurred: the corresponding reaction execution will result in a new `out(engine_trigger)` operation that keeps the chemical engine running. Finally, a new tuple `execution(law(IL,_,OL),Time)` is inserted so that the set of reactions devoted to chemical-law execution can be activated.

The actual implementation of the Gillespie's algorithm regarding the choice of the chemical law to be executed is embedded in the `chooseLaw` predicate, whose implementation is as follows:

```
chooseLaw(Law,Rtot):-
          rd(laws(LL)),
          semanticMatchAll(LL,NL,Rtot),
          not(Rtot==0),
          sortLaws(NL,SL),
          rand_float(Tau),
          chooseGillespie(SL,Rtot,Tau,Law).
```

After retrieving the list `LL` of the chemical laws defined for the tuple centre, `semanticMatchAll` returns the list `NL` of enabled chemical laws and the corresponding overall rate `Rtot`, computed as the sum of the global rate of every enabled law. To this end, predicate `semanticMatchAll` relies on predicate `retrieve(+SemanticTemplateList, -SemanticTupleList,-MatchFactor)` already described (properly extended to deal with lists of semantic tuples and templates).

The chemical law to be executed is actually chosen via the `chooseGillespie` predicate if `Rtot` > 0, i.e. if there are enabled chemical laws. This choice is driven by a probabilistic process: given n chemical laws and their global rates $r_1, ..., r_n$, the probability for law i to be chosen is defined as r_i/R, where $R = \sum_i r_i$. Consequently, law selection is simply driven by drawing a random number between 0 and 1 and choosing a law according to the probability distribution of the enabled laws.

V. Related Work

Coordination models. The issue we face in this article can be framed as the problem of finding the proper coordination model for enabling and ruling interactions of pervasive services. Coordination models generated by the archetypal LINDA model [15], which simply provides for a blackboard with associative matching for mediating component interactions through insertion/retrieval of tuples. A radical change is instead the idea of engineering the coordination space of a distributed system by some policy "inside" the tuple spaces as proposed here, following the pioneer works of e.g. TuCSoN [24]—in fact, as shown in this paper, TuCSoN can be used as a low-level virtual platform for enacting the chemical tuple-space model. As already mentioned, the work presented in this article is based on [31], which was extended in [32] to deal with self-composition of services, a concept that we can support in our framework though it is not addressed in this paper.

Chemistry has been proposed as an inspiration for several works in distributed computing and coordination over many years, like in the Gamma language [10] and the chemical abstract machine [6], which lead to the definition of some general-purpose architectures [18]. Although these models already show the potential of structuring coordination policies in terms of chemical-like rewriting rules, we observe that they do not bring the chemical metaphor to its full realisation as we do here, as they do not exploit chemical stochastic rates.

Situatedness. In many proposals for pervasive computing environments and middleware infrastructures, the idea of "situatedness" has been promoted by the adoption of shared virtual spaces for services and components interactions. Gaia [28] introduces the concept of active spaces, a middleware infrastructure enacting distributed active blackboard spaces for service interactions. Later on, a number of proposals have extended upon Gaia, to enforce dynamic semantic pattern-matching for service composition and discovery [14] or access to contextual information [12]. Other related approaches include: X-KLAIM [8], proposing tuple space as a coordination media with the explicit use of localities for accessing data or computational resources; Egospaces [19], exploiting a network of tuple spaces to enable location-dependent interactions across components; LIME [27], proposing tuples spaces that temporarily merge based on network proximity, to facilitate dynamic interactions and exchange of information across mobile devices; and TOTA [20], enacting computational gradients for self-awareness in mobile networks. Our model shares the idea of conceiving components as "living" and interacting in a shared spatial

substrate (of tuple spaces) where they can automatically discover and interact with one another. Yet, our aim is broader, namely, to dynamically and systemically enforce situatedness, service interaction, and data management with a simple language of chemical reactions, and most importantly, enacting an ecological behaviour thanks to the support of diversity in the long term.

Self-organisation. Several recent works exploit the lessons of adaptive self-organising natural and social systems to enforce self-awareness, self-adaptivity, and self-management features in distributed and pervasive computing systems. At the level of interaction models, these proposals typically take the form of specific nature- and socially inspired interaction mechanisms [3] (e.g. pheromones [26] or virtual fields [20]), enforced either at the level of component modelling or via specific middleware-level mechanisms. We believe our framework integrates and improves these works in two main directions: *(i)* it tries to identify an interaction model that is able to represent and subsume the diverse nature-inspired mechanisms via a unifying self-adaptive abstraction (i.e. the semantics chemical reactions); *(ii)* the "ecological" approach we undertake goes beyond most of the current studies that limit to ensembles of homogeneous components, supporting the vision of novel pervasive and Internet scenarios as a sort of cyber-organisms [1].

VI. Roadmap and Conclusion

In this paper we described research and development challenges in the implementation of the chemical tuple space model in TuCSoN. These are routed in two basic dimensions, which are mostly – but not entirely – orthogonal.

On the one hand, the basic tuple centre model is to be extended to handle semantic matching, which we support by the following ingredients: *(i)* an OWL ontology (a set of definitions of concepts) stored into a tuple centre which grounds semantic matching; *(ii)* tuples (other than syntactic as usual) can be semantic, describing an individual of the application domain (along with the concept it belongs to and the individuals it is linked to through roles); and *(iii)* a matching function implemented so as to check whether a tuple is the instance of a concept, returning the corresponding match factor.

On the other hand, the coordination specification for the tuple centre should act as a sort of "online chemical simulator", evolving the concentration of tuples over time using the same stochastic model of chemistry [16], so as to reuse existing natural and artificial chemical systems (like prey-predator equations); at each step of the process: *(i)* the reaction rates of all the chemical laws are computed, *(ii)* one is probabilistically selected and then executed, *(iii)* the next step of the process is triggered after an exponentially distributed time interval, according to the Markov property.

The path towards a fully featured and working infrastructure has been paved, but further research and development is required to tune several aspects:

Match factor. Studying suitable fuzzy matching techniques is currently a rather hot research topic, e.g. in the Semantic Web context. Our current support trades off simplicity for expressive power, but we plan to extend it using some more complete approach and in light of the application to selected cases—e.g. fully relying on [9].

Performance. The problem of performance was not considered yet, but will be subject of our future investigation. Possible bottlenecks include the chemical model and its implementation as a ReSpecT program, but also semantic retrieval, which is seemingly slower than standard syntactic one. We still observe that in many scenarios of pervasive computing this is not a key issue.

Chemical language. Developing a suitable language for semantic chemical laws is a rather challenging issue. The design described in this paper supports limited forms of service interactions that will likely be extended in the future. For instance, a general law $X + Y \rightarrow Z$ is meant to combine two individuals into a new one, hence the chemical language should be able to express into Z *how* the semantic templates X and Y should combine—aggregation, contextualisation, and other related patterns of self-organising pervasive systems are to be handled at this level.

Application cases. The model, and correspondingly the implementation of the infrastructure, are necessarily to be tuned after evaluation of selected use cases can be performed. Accordingly, the current version of the infrastructure is meant to be a prototype over which initial sperimentation can be performed. A main application scenario we will considered for actual implementation is a general purpose pervasive display infrastructure.

References

[1] G. Agha. Computing in pervasive cyberspace. *Commun. ACM*, 51(1):68–70, 2008.

[2] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[3] O. Babaoglu, G. Canright, A. Deutsch, G. A. D. Caro, F. Ducatelle, L. M. Gambardella, N. Ganguly, M. Jelasity, R. Montemanni, A. Montresor, and T. Urnes. Design patterns from biology for distributed computing. *ACM Trans. Auton. Adapt. Syst.*, 1(1):26–66, 2006.

[4] A. Bandara, T. R. Payne, D. D. Roure, N. Gibbins, and T. Lewis. A pragmatic approach for the semantic description and matching of pervasive resources. In *Advances in Grid and Pervasive Computing*, volume 5036 of *LNCS*, pages 434–446. Springer, 2008.

[5] A. P. Barros and M. Dumas. The rise of web service ecosystems. *IT Professional*, 8(5):31–37, 2006.

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

[7] A. A. Berryman. The origins and evolution of predator-prey theory. *Ecology*, 73(5):1530–1535, October 1992.

[8] L. Bettini, R. De Nicola, and M. Loreti. Implementing Mobile and Distributed Applications in X-Klaim. *Scalable Computing: Practice and Experience, Special Issue: Software Agent Mobility*, 7(4):13–35, 2006.

[9] F. Bobillo and U. Straccia. fuzzyDL: An expressive fuzzy description logic reasoner. In *2008 International Conference on Fuzzy Systems (FUZZ-08)*, pages 923–930. IEEE Computer Society, 2008.

[10] J.-P. Bonâtre and D. Le Métayer. Gamma and the chemical reaction model: Ten years after. In *Coordination Programming*, pages 3–41. Imperial College Press London, UK, 1996.

[11] S. Camazine, J.-L. Deneubourg, N. R. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau. *Self-Organization in Biological Systems*. Princeton Studies in Complexity. Princeton University Press, Princeton, NJ, USA, 2001.

[12] P. D. Costa, G. Guizzardi, J. P. A. Almeida, L. F. Pires, and M. van Sinderen. Situations in conceptual modeling of context. In *Tenth IEEE International Enterprise Distributed Object Computing Conference (EDOC 2006), 16-20 October 2006, Hong Kong, China, Workshops*, page 6. IEEE Computer Society, 2006.

[13] A. Ferscha, A. Riener, M. Hechinger, and H. Schmitzberger. Building pervasive display landscapes with stick-on interfaces. In *CHI Workshop on Information Visualization and Interaction Techniques*, April 2006.

[14] 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, *Coordination Languages and Models*, volume 5521 of *LNCS*, pages 66–85. Springer-Verlag, June 2009. 11th International Conference (COORDINATION 2009), Lisbon, Portugal, June 2009. Proceedings.

[15] D. Gelernter. Generative communication in linda. *ACM Trans. Program. Lang. Syst.*, 7(1):80–112, 1985.

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

[17] I. Horrocks, P. F. Patel-Schneider, and F. V. Harmelen. From shiq and rdf to owl: The making of a web ontology language. *Journal of Web Semantics*, 1:2003, 2003.

[18] P. Inverardi and A. L. Wolf. Formal specification and analysis of software architectures using the chemical abstract machine model. *IEEE Trans. Software Eng.*, 21(4):373–386, 1995.

[19] C. Julien and G.-C. Roman. Egospaces: Facilitating rapid development of context-aware mobile applications. *IEEE Trans. Software Eng.*, 32(5):281–298, 2006.

[20] M. Mamei and F. Zambonelli. Programming pervasive and mobile computing applications: the TOTA approach. *ACM Trans. Software Engineering and Methodology*, 18(4), 2009.

[21] E. Nardini, M. Viroli, and E. Panzavolta. Coordination in open and dynamic environments with TuCSoN semantic tuple centres. In S. Y. Shin, S. Ossowski, M. Schumacher, M. Palakal, C.-C. Hung, and D. Shin, editors, *25th Annual ACM Symposium on Applied Computing (SAC 2010)*, volume III, pages 2037–2044, Sierre, Switzerland, 22–26 Mar. 2010. ACM. Awarded as Best Paper.

[22] L. j. b. Nixon, E. Simperl, R. Krummenacher, and F. Martin-recuerda. Tuplespace-based computing for the semantic web: A survey of the state-of-the-art. *Knowl. Eng. Rev.*, 23(2):181–212, 2008.

[23] A. Omicini. Formal ReSpecT in the A&A perspective. *Electronic Notes in Theoretical Computer Sciences*, 175(2):97–117, June 2007.

[24] A. Omicini and F. Zambonelli. Coordination for Internet application development. *Autonomous Agents and Multi-Agent Systems*, 2(3):251–269, Sept. 1999.

[25] M. Paolucci, T. Kawamura, T. R. Payne, and K. P. Sycara. Semantic matching of web services capabilities. In *International Semantic Web Conference*, volume 2342 of *LNCS*, pages 333–347. Springer, 2002.

[26] H. V. D. Parunak, S. Brueckner, and J. Sauter. Digital pheromone mechanisms for coordination of unmanned vehicles. In *Autonomous Agents and Multiagent Systems (AAMAS 2002)*, volume 1, pages 449–450. ACM, 15–19 July 2002.

[27] G. P. Picco, A. L. Murphy, and G.-C. Roman. LIME: Linda meets mobility. In *The 1999 International Conference on Software Engineering (ICSE'99)*, pages 368–377. ACM, 1999. May 16-22, Los Angeles (CA), USA.

[28] M. Román, C. K. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt. Gaia: a middleware platform for active spaces. *Mobile Computing and Communications Review*, 6(4):65–67, 2002.

[29] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *J. Web Sem.*, 5(2):51–53, 2007.

[30] M. Ulieru and S. Grobbelaar. Engineering industrial ecosystems in a networked world. In *5th IEEE International Conference on Industrial Informatics*, pages 1–7. IEEE Press, June 2007.

[31] M. Viroli and M. Casadei. Biochemical tuple spaces for self-organising coordination. In J. Field and V. T. Vasconcelos, editors, *Coordination Languages and Models*, volume 5521 of *LNCS*, pages 143–162. Springer-Verlag, June 2009. 11th International Conference (COORDINATION 2009), Lisbon, Portugal, June 2009. Proceedings.

[32] M. Viroli and M. Casadei. Chemical-inspired self-composition of competing services. In S. Y. Shin, S. Ossowski, M. Schumacher, M. Palakal, C.-C. Hung, and D. Shin, editors, *25th Annual ACM Symposium on Applied Computing (SAC 2010)*, volume III, pages 2029–2036, Sierre, Switzerland, 22–26 Mar. 2010. ACM.

[33] M. Viroli, M. Casadei, and A. Omicini. A framework for modelling and implementing self-organising coordination. In *24th Annual ACM Symposium on Applied Computing (SAC 2009)*, volume III, pages 1353–1360. ACM, 8–12 Mar. 2009.

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

[35] F. Zambonelli and M. Viroli. Architecture and metaphors for eternally adaptive service ecosystems. In *IDC'08*, volume 162/2008 of *Studies in Computational Intelligence*, pages 23–32. Springer Berlin / Heidelberg, September 2008.