Computational Fields meet Augmented Reality: Perspectives and Challenges

Danilo Pianini, Angelo Croatti, Alessandro Ricci, Mirko Viroli
Alma Mater Studiorum – Università di Bologna, Dept. of Computer Science and Engineering (DISI)
{danilo.pianini, a.croatti, a.ricci, mirko.viroli}@unibo.it

***Abstract*—Recently, two different techniques emerged that are tailored to environments pervaded of computational devices. On the one hand, aggregate programming, and especially computational fields-based programming, is a promising abstraction for coordinating the activities of multiple situated devices. On the other hand, augmented reality is emerging as new means of interaction with both software and physical entities.**

We note that both computational fields and augmented reality are tightly bound to the physical world, and that they both enrich it, with collective computation and augmented information respectively. This work presents an initial analysis of possible future research directions that involve these techniques, discussing some possible ways of integrating them.

I. Introduction

The pervasive availability of devices able to communicate and equipped with decent computational capabilities is leading to a new world where computation is everywhere. This kind of world is now studied under a plethora of names, such as "Internet of Things", "Pervasive Computing", "Smart Cities", and so on. One of the research lines that aims at providing suitable computational abstractions for this kind of applications is rooted around the notion of computational field. A *computational field* is a physically distributed data structure, seen as the atomic data computationally manipulated. It is the conceptual abstraction for the design and the development of distributed and pervasive systems, where the focus is centred in computations and coordination of aggregates of devices instead of their individual behaviour. Computational fields lead to an innovative way to design and develop software systems.

At the same time, miniaturisation of displays and new achievements in disciplines such as computer vision are leading towards novel means of interaction with software, that could be used for "augmenting" the physical world, namely, building a whole augmented reality. In the paper we explore some (incremental) ways to integrate augmented reality (AR) and programming based on computational fields (CFs). In our vision, AR could have a remarkable impact on the development and testing of software systems designed as aggregates, and at the same time aggregate programming and computational fields provides an appealing set of abstractions for building advanced AR-aware applications. In this paper we explore the integrated use of AR technologies and languages for aggregate computing (computational fields oriented programming languages), with the purpose to provide an effective way of "visualizing" computational fields, as well as exploiting AR in order to increase the range of possible interaction means between users and software. The goal of this paper is to shed light to an appealing research area that to the best of our knowledge has not yet been explored.

The remainder of this paper is organised as follows. Section II introduces Augmented Reality. Section III presents the concepts of aggregate programming and computational fields. In Section IV different types of integration between computational fields and augmented reality are discussed. In Section V some programming examples are provided for such integration types. Finally, Section VI draws conclusions.

II. Augmented Reality (AR)

Nowadays, Augmented Reality (AR) is a well-known technology that can be informally defined as a way to join *virtual elements* (intended as suitable representations of data and information) with the real world. A more accurate definition has been further proposed by R. T. Azuma in [1] where AR is defined as a technology that allows the user to see the real world, with virtual objects superimposed upon or mixed with the real world. Main properties of AR technology are: (i) Seamless interaction between real and virtual environments; (ii) Physical world's enhancing, adding information; (iii) Real-time coupling between the real-world perception and the augmented information layer.

A. Supports and Frameworks for Augmented Reality

The union of virtual elements with the real world (more precisely, the overlapping of virtual elements on the user's real world perception) can be obtained exploiting a wide spectrum of available (mobile) devices. In particular, to exploit AR technologies, the feature to perceive the real-time state of the world (e.g. exploiting a camera) is enough to provide to the user an *augmented view* of the real-world.

Nevertheless, recent developments in wearable computing have led to the availability of new interesting mobile devices like AR-glasses (e.g Epson Moverio BT-200) [2] [3]. Most of such devices are classified as "see-through" devices; thus they offer the possibility to directly perceive the real-world, without necessarily using a camera, superimposing virtual elements directly on the lenses of AR-glasses. This way, the user can be introduced in a kind of *augmented world*, where the union of virtual elements with the real world can be completely seamless.

On the side of existing frameworks, proposed to enable the use of AR technologies, we can find many solutions

978-1-4673-8439-1/15 $31.00 © 2015 IEEE
DOI 10.1109/SASOW.2015.18

that facilitate software application development. For example, general-purpose frameworks like *LayAR*[1] and *Wikitude*[2] provides a kind of library with predefined functionalities, mainly oriented to make it easier the superimposing of augmented data and information within the user's field of view. The current state of these technologies is limited to offer an efficient way (1) to recognize markers and keep coupled over them an augmented representations of information and (2) to associate information/geometries to located Points-of-interest (POIs), showing this information when the user moves itself into the POI's perception range. Unfortunately, most of these frameworks seem not able to provide an adequate support to the manipulation of dynamic augmented elements. For instance, considering an augmented element that represent the smart view of a dynamic computational entity, if this entity changes its internal status and has to update its view, they cannot update a part (i.e., a detail) of the augmented element but need to create a new geometry to replace the old one. Moreover, these frameworks often do not enable the dynamic creations for geometries but all geometries for all possible states must be generated offline and provided to the application in the development phase.

B. Different degrees of augmentation

The use of AR in modern software systems, as a new way to conceive and design (a part of) the user interface, provides a wide range of new potentialities for designers. Data and information (1) can be presented to the user in a completely different way and (2) the fact that data are continuously (real-time) coupled with reality perceived by the user may bring to the emergence of new (implicit) interaction mechanisms between user and the computational system.

Currently, AR can be conceived through different degrees of abstraction, from a weak overlapping of some messages to a more immersive and seamless union of virtual and physical reality. Principally, we can define three different ways to realize augmented/mixed reality:

1) Overlaying to the real world some augmented information (messages or generic textual data and/or 2D images) completely unrelated to the position/shape of real elements in the current user field of view;
2) Superimposing information dynamically associated to some elements properly recognized within the user field of view; or finally
3) Superimposing 2D/3D elements linked to the position of real world elements or related to specific markers.

Each degree of abstraction proposes a different way to use AR technologies into software applications. In particular, in this paper we focus on two possible usages of AR technologies: (i) we want to bring into the user's perception situated information generated by computational fields – for instance, showing to the user a representation of the computational field through an hands-free oriented system – and (ii) we also want to enrich the computational field approach providing an innovative support based on augmented/mixed reality with the purpose to enable brand new usage scenarios for computational fields.

[1] https://www.layar.com
[2] https://www.wikitude.com

III. Aggregate Programming and Computational Fields (CF)

Aggregate programming is a programming model inspired by the observation that in many cases the designer of a large-scale system are much less concerned with individual devices than with the services provided by the collection of devices as a whole. Typical device-centric programming languages, however, force a programmer to focus on individual devices and their interactions. As a consequence, several different aspects of a distributed system typically end up entangled together: effectiveness and reliability of communications, coordination in face of changes and failures, and composition of behaviours across different devices and regions. This makes it very difficult to effectively design, debug, maintain, and compose complex distributed applications.

Aggregate programming generally attempts to address this problem by providing composable abstractions separating these aspects:

1) device-to-device communication is typically made entirely implicit, with higher-level abstractions for controlling efficiency/robustness trade-offs;
2) distributed coordination methods are encapsulated as aggregate-level operations (e.g., measuring distance from a region, spreading a value by gossip, sampling a collection of sensors at a certain resolution in space and time); and
3) the overall system is specified by composing aggregate-level operations, and this specification is then transformed into a complete distributed implementation by a suitable mapping.

Many aggregate programming applications involve collections of spatially embedded devices [4], where geometric operations and information flow provide a useful source of aggregate-level abstractions. From this, a large number of different methods have been developed, including such diverse approaches as abstract graph processing (e.g., [5]), declarative logic (e.g., [6]), map-reduce (e.g., [7]), streaming databases (e.g., [8]), and knowledge-based ensembles (e.g., [9])—for a detailed review, see [4]. One of the most successful abstraction, however, was based on viewing the collection of devices as an approximation of a continuous field (e.g. in [10], [11], [12], [13], [14], [15], [16]).

Recently, a minimal core calculus of computational fields [17], has been derived from the commonalities of these methods. The field calculus is based on the notion of a *computational field*—a map from devices comprising the system to (possibly structured) values, which is treated as unifying first-class abstraction to model system evolution and environment dynamics. Based on such calculus, a new language – Protelis – has been developed that allows for practical experiments with such abstractions, both in simulation and real devices [18].

IV. Augmented Fields

In this section, we analyse how AR and CF may combine. We believe that such integration could happen at different levels, with different depths, and the way it is performed may impact both the kind of applications that the system may support and the software engineering methodology that allows to build such applications.

A. AR as visual interface for CF programs

As a first step, we consider AR as a visualisation interface for a CF program. CF are agnostic to the specific technology used to expose the results of the computation, however, most of the case studies available in literature involve situated devices and users. Not much has been written with respect to the visualisation technology by which such CF programs deliver information to users, many examples suppose the users be equipped with one or more handheld devices with a small monitor. We believe that, for such situated programs, an AR visualisation interface would be a very natural way to expose the computation results. In fact, it would provide an immediate feedback about the current situation, with no need to proactively check a handheld's screen.

As an example, imagine a crowd detection system such as the one presented in [19]. This software relies on Computational Fields in order to identify areas where a crowd is dense up to the point that it becomes dangerous, and provides feedback to users so that they can avoid such overly-populated zones. Possible user notification methods range from a colour gradient on a map onto the handheld display to a vibration when the speed vector of the user is heading towards the dangerous areas. The former would provide a richer information, but requires proactive behaviour from the user. The second can be performed with no explicit user action, but does not provide a complete information about the surroundings. A possible AR display could take the best of both, providing the user full information without requiring any proactive behaviour. It could, for instance, colour the field of view of the user with different tones depending on the measured crowd level. A pictorial representation of such view is depicted in Figure 1.

Augmented reality visors can be of use also for testing programs based on computational fields. The common practice when developing programs based on computational fields is to run simulations to test the software behaviour in a controlled setup. Simulations are surely key in the development process, but in general they cannot provide the full spectrum of information that is needed prior to the actual deployment. Simulations in CF programs are much like unit testing in classic software development: necessary, but not sufficient to guarantee that things will run smoothly on a real world setup. In fact, any simulation operates upon a model of the world, and abstracts some of its details. A serie of real-world tests before actual deployment would be much like the beta-testing phase of a classic software product. Augmented reality visualisation can help the developer by providing a real-time, immediately understandable visualisation of computational fields. A possible form of visualisation is pictured in Figure 2.

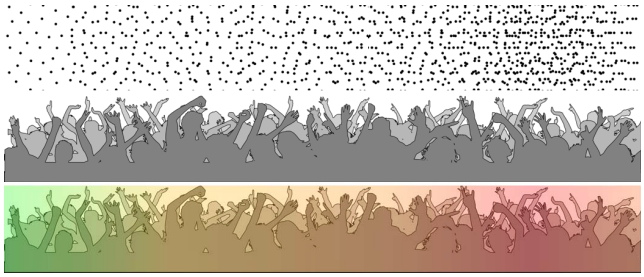

Fig. 1. A possible augmented reality powered display for a crowd warning application. User perception of a crowd may be incomplete or wrong due to the single user's non-privileged point of view. On top, the real distribution of people is depicted. The middle image is a pictorial representation of the user's non-enhanced field of view, while the image on bottom is the same field of view with an augmented reality visor enabled.

B. Interaction in CF applications through AR

The step immediately following the visualisation is the interaction with a CF program. CF languages are normally agnostic to the interaction method. One of the possible interaction models is the presence of sensors (variables whose values could change because of external events, e.g. the GPS position of the device) and actuators (variables that can be set and may influence the external world, e.g. making a smartphone vibrate or visualise some data on smart glasses). If we adopt this view, we have previously discussed the augmented reality as actuator, and here we reason about the augmented reality as sensor.

We see two possible forms of interaction: a weak one, in which AR is used to improve, ease and make more natural the normal interaction with a CF program; and a strong one, in which AR may inject totally new information, absent in the real world. The "weak" form of interaction is basically about generating part of the data a CF program computes upon by means of interacting through AR devices. The same kind

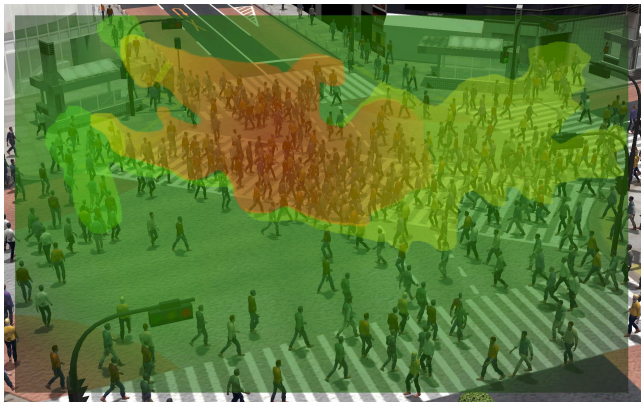

Fig. 2. A possible augmented reality powered display could allow deeper understanding of the actual shape of a computational field.

of information could have been generated differently, so the big leap here is just in the immediateness of interaction. It is worth noting, however, that such interactivity would enable much more effective development practices if compared to a simpler "visualisation only" user interface. In fact, the tester may now perturb the fields while the application is running: she would be able to better understand what may happen in otherwise difficult to reproduce situations, or to make sure the system is resilient to failures or malfunctioning, and how quickly it re-stabilises to a correct situation.

The "strong" interaction form is inspired by the so-called "mirror world" approach [20], in which the real world is enriched by an augmented layer where the entities of the world are reflected (along with their status), and where other entities, completely virtual, may live. The mirror world vision is not very distant from a "field of entities" that comprises both real world entities and virtual entities. The latter entities, which may be injected or removed, and which users may interact with, introduce an abstraction that is not trivially expressible using computational fields. Normally, in fact, computational fields can be seen as data structures living in the whole space-time continuum [21], and whose value is sampled in those positions where actual devices are situated. For this reason, the higher the density of devices, the closer is a computational field program to its "continuous" version. An augmented "mirror world" would introduce a novelty in such view: new entities may be spawned that belong to the virtual layer alone, without any counter-part in the real world, where physical devices responsible for "sampling" the world are situated. The most classic approach, as a consequence, would be viable only when new elements of the virtual layer could be added where a real device is present, or in the situation in which the device density is so high that we are guaranteed that the augmented entity injected can always be kept in existence by one or more devices.

Computational fields, however, can be exploited to deal with such virtual entities in a better way. We have previously supposed that the mirror world could be modelled as a field of entities, and that guaranteed that every computational device could have a doppelganger in the virtual layer. The field, however, could be for instance a field of mappings between entities and their location. This kind of field would be able to host in a sparse network of computational devices the idea of mirror world, where the virtual layer hosts both a representation of any real-world entity and some entities that do not have a real-world counterpart. Clearly, such possibility must be considered since the early design stage of the CF program: "strong" interaction requires the CF application be designed appropriately, and it is not a sort of free lunch like the integration of an AR UI or a "weak" interaction form.

C. CF as enabling technology for AR applications

In the previous section, we integrated AR and CF so deeply that CF applications must have considered such integration into account in the first place. Since we need to take AR into account in our CF applications if we want to reproduce ideas such as the mirror world, our next step will be to try to flip our point of view, and instead of exploring the possibilities offered by an augmentation of computational fields, we try to understand if and how the CF abstraction could be a solid foundation for AR applications.

The idea of representing the augmented "mirror world" as a sort of field living over the existing devices sounds naively interesting. However, as we have discussed previously, this simple approach has problems unless we suppose that the mirror world exists only where the field can be sampled (namely, where devices are located). This simplification could be acceptable for a wide range of applications, but an application like the ghost game [20] could not be created, since they feature what we have called "strong" interaction.

A different approach however could be used to deal with virtual entities existing in points where no device can sample them. The basic strategy has already been described: switch from a simple field of mirrored entities to a field of entities with location. Languages supporting computational fields as first class abstraction, such as Protelis [18], provide support for enclosing the creation and maintenance of such structures into reusable building blocks.

Although useful, however, reusable building blocks are not the only features that make CF abstractions appealing for AR applications. In recent works higher order functions have been exploited in computational fields to provide support for mobile code [22], and initial steps towards a core computational fields based framework for distributed applications have been taken [19]. We believe that such approach may provide a solid foundation for augmented-reality applications, providing abilities to inject, execute, and update multiple augmented reality applications, relying on computational fields to decide which devices should participate an application.

V. Examples

In this section, we make our previous thoughts more concrete, providing some code examples in Protelis of possible integration between AR and CF. The reasons why we use Protelis instead of other languages, such as Proto [10], are mainly the support for higher order functions, that enable mobile code, and the interoperability with a widespread programming language, Java. Java already possesses a rich collection of libraries, and is likely to be target of further developments of AR libraries, due to being the language used by most applications of one of the major mobile platforms, Google Android. As a consequence, we believe that Java interoperability makes Protelis the current best candidate among computational fields oriented languages for supporting AR.

In the remainder we will follow the progression of Section IV, first discussing how to visualise fields, then presenting a possible interaction model, and concluding with a CF-based distributed virtual machine.

A. Display computational fields in AR

There are different approaches when it comes to attaching a UI to a field calculus program. One possibility is to allow

```
import protelis:experimental:ieeeiot:buildingblocks

/*
 * Given an estimation of the probability that an user is
 * running the crowd estimation app, returns the local
 * density
 */
def localDensity(probability, radius) {
  average(
    managementRegions(radius * 2, () -> { nbrRange }),
    densityEst(probability, radius)
  )
}
let res = localDensity(0.1, 30);
self.putEnvironmentVariable("res", res);
res
```

Fig. 3. Simple Protelis program that computes the crowd density in the surroundings and prepares the output for being visualised by an AR system. For the sake of conciseness, we imported a module developed for another work [19].

```
import protelis:experimental:ieeeiot:buildingblocks
def localDensity(probability, radius) { ... }
def summarizeDensity(prob, radius) {
  c(0, (loc, other) -> { loc.union(other) },
  [[self.getDevicePosition(), localDensity(prob, radius)]],
  [])
}
let res = summarizeDensity(0.1, 30);
self.putEnvironmentVariable("res", res);
res
```

Fig. 4. Protelis program that exposes to the external world a tuple of mappings between coordinates and perceived crowd density. For the sake of conciseness, we imported a module developed for another work [19], in particular we reuse the "converge" building block `C` [23].

the UI to inspect the status of a CF program. The UI must be tailored to a specific language, and the language should allow for inspection. The advantage of such approach is in that no change should be made to a working CF program to visualise it in augmented reality, but the downside is that a stand-alone application is required between the CF program and the AR technologies that is aware of the program structure. We believe that this strategy could be effective for development and debugging purpose, but not optimal for general use. The alternative is that the CF program is aware of the existence of UI. If it is, then it could be aware of it at various levels. The most natural support for providing output in computational fields, besides the result of the local program itself, is the usage of actuators. The program can write data on such "actuators", that are in turn responsible of using it. In Protelis, for instance, actuators are accessed through the same mechanisms that allows for registering global variables by calls to `self.putEnvironmentVariable(name, value)`. Such variables are exported in a device-dependent way, and can be used by other processes on the same machine. As an example, in Figure 3 we show a possible implementation of a program that outputs the local density of people, given a probability that the users around are running the crowd detection application and a range of interest.

This kind of implementation provides just a local value. We may want, in general, to have values for the places around us,

```
def entitiesInRange(name, range) {
  rep (known <- []) {
    unionHood PlusSelf(nbr(
      known.union(
        unionHood PlusSelf(
          // Entities injected locally are echoed in the neighbourhood
          mux(nbr(self.hasEnvironmentVariable(name))) {
            let ents = self.getEnvironmentVariable(name);
            if(self.
              hasEnvironmentVariable(name)) {
              // map from to a field of pairs position/entity
              ents.map(self, (e) -> {[
                let epos = e.getPosition();
                if (self.getDevicePosition()
                  .getDistanceTo(epos) < range) {
                  epos
                } else {
                  [NaN, NaN]
                }, e]})
            } else {
              []
            }
          }
        )
      )
    )).filter(self,
        (t) -> {t.get(0).getDistanceTo(
          self.getDevicePosition()) < range}
    )
  }
}
```

Fig. 5. Protelis program that materialises and keeps in sync a purely virtual entity onto nearby devices (within `range` distance from the virtual entity position). In this code, entities are supposed to be represented by Java objects, provided with a `getPosition()` method.

in order to display them. The need of summarising information from a part of the network back to a source has been already identified as a fundamental operation in other works [22], [23], [19], and CF languages offer nice support for building such operations. In Figure 4 we show a possible Protelis implementation relying on existing operators, that outputs a tuple of pairs of coordinates and perceived crowd densities. Some CF languages may support calling the AR API directly. In these cases, then the CF program could take care of dealing with the visualisation directly, rather than just reporting a result for an external process to be used.

B. Interaction with AR

As mentioned in Section IV, the most natural way to have weak interaction is by injecting AR-produced data as sensors. Sensors data is accessed in Protelis with a function which is dual to actuator access: `self.getEnvironmentVariable(name)`, that returns the value associated with sensor `name`. In weak interaction, input from AR devices is not distinguishable from input coming from any other source from the point of view of computational fields. Because of that, management of such information is not particularly difficult, nor it requires AR-tailored solutions. This sounds appealing, but the downside is that it only provides limited functionality.

A way to support strong interaction, as discussed in Section IV requires the ability of maintaining a field that associates augmented objects that exist only in the virtual layer with their position. In Figure 5 we exemplify a possible (rather

raw) solution that could materialise and keep into existence objects in virtual layer, modelled as a field of locations and entities. This code is not to be considered stable or definitive, but rather is a stub for more reliable implementations.

A more elaborated and stable solution may make use of a sort of "distributed virtual machine" such as the one proposed in [19]. This kind of machine allows for injection, scheduling, and maintaining of computational fields programs into running machines. Such a solution could be easily adapted for injection and maintenance of augmented entities.

VI. Conclusion

In this paper we have discussed a possible convergence between computational fields and augmented reality. Both technologies aim at enriching the physical world, but they tackle the type of enrichment in a different way: computational fields exploit distributed computing abilities to perform collective computation; augmented reality enriches the perceived world with additional information, presents it to the user in a natural way, and possibly takes input from her. Though our analysis is currently at an early stage, we reckon that there is space for these technologies to be fruitfully used together. We have showed some possible approaches, ranging from the usage of an augmented reality user interface for applications and developers to the possibility of building a distributed AR-aware virtual machine based on computational fields. All of them could be easily implemented in existing computational fields infrastructures, and we presented some code examples. Obviously, our hypothesis that computational fields are a particularly suitable framework for augmented reality appliances require experimental confirmation. With our current knowledge, however, our position is that such convergence is a promising research path, and an engineering issue worth investigation.

References

[1] R. Azuma, "A survey of augmented reality," vol. 6, no. 4, pp. 355–385, 1997.

[2] T. Starner, S. Mann, B. Rhodes, J. Levine, J. Healey, D. Kirsch, R. W. Picard, and A. Pentland, "Augmented reality through wearable computing," 1997.

[3] L. Avila and M. Bailey, "Advanced display technologies," *Computer Graphics and Applications, IEEE*, vol. 35, no. 1, pp. 96–97, 2015.

[4] J. Beal, S. Dulman, K. Usbeck, M. Viroli, and N. Correll, "Organizing the aggregate: Languages for spatial computing," in *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments*, M. Mernik, Ed. IGI Global, 2013, ch. 16, pp. 436–501.

[5] R. Gummadi, O. Gnawali, and R. Govindan, "Macro-programming wireless sensor networks using kairos." in *Distributed Computing in Sensor Systems (DCOSS)*, 2005, pp. 126–140.

[6] M. P. Ashley-Rollman, S. C. Goldstein, P. Lee, T. C. Mowry, and P. Pillai, "Meld: A declarative approach to programming ensembles," in *IEEE Intelligent Robots and Systems (IROS)*, 2007, pp. 2794–2800.

[7] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[8] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, "Tinydb: An acqusitional query processing system for sensor networks," in *ACM TODS*, 2005.

[9] R. D. Nicola, M. Loreti, R. Pugliese, and F. Tiezzi, "A formal approach to autonomic systems programming: The scel language," *ACM Trans. Auton. Adapt. Syst.*, vol. 9, no. 2, pp. 7:1–7:29, Jul. 2014. [Online]. Available: http://doi.acm.org/10.1145/2619998

[10] J. Beal and J. Bachrach, "Infrastructure for engineered emergence in sensor/actuator networks," *IEEE Intelligent Systems*, vol. 21, pp. 10–19, March/April 2006.

[11] M. Mamei and F. Zambonelli, "Programming pervasive and mobile computing applications: The tota approach," *ACM Trans. on Software Engineering Methodologies*, vol. 18, no. 4, pp. 1–56, 2009.

[12] R. Newton and M. Welsh, "Region streams: Functional macroprogramming for sensor networks," in *First International Workshop on Data Management for Sensor Networks (DMSN)*, Aug. 2004, pp. 78–87.

[13] M. Viroli, M. Casadei, S. Montagna, and F. Zambonelli, "Spatial coordination of pervasive services through chemical-inspired tuple spaces," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 6, no. 2, pp. 14:1 – 14:24, June 2011.

[14] M. Mamei and F. Zambonelli, "Self-maintained distributed tuples for field-based coordination in dynamic networks," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 4, pp. 427–443, 2006.

[15] R. Nagpal, "Programmable self-assembly: Constructing global shape using biologically-inspired local interactions and origami mathematics," Ph.D. dissertation, MIT, Cambridge, MA, USA, 2001.

[16] D. Yamins, "A theory of local-to-global algorithms for one-dimensional spatial multi-agent systems," Ph.D. dissertation, Harvard, Cambridge, MA, USA, December 2007.

[17] M. Viroli, F. Damiani, and J. Beal, "A calculus of computational fields," in *Advances in Service-Oriented and Cloud Computing*, ser. Communications in Computer and Information Sci., C. Canal and M. Villari, Eds. Springer Berlin Heidelberg, 2013, vol. 393, pp. 114–128.

[18] D. Pianini, M. Viroli, and J. Beal, "Protelis: Practical aggregate programming," in *SAC '15: Proceedings of the 30th Annual ACM Symposium on Applied Computing*. Salamanca, Spain: ACM, 2015.

[19] J. Beal, D. Pianini, and M. Viroli, "Aggregate programming for the internet of things," *IEEE Computer*, To Appear on Sept. 2015.

[20] A. Ricci, M. Piunti, L. Tummolini, and C. Castelfranchi, "The mirror world: Preparing for mixed-reality living," *IEEE Pervasive Computing*, vol. 14, no. 2, pp. 60–63, 2015. [Online]. Available: http://dx.doi.org/10.1109/MPRV.2015.44

[21] J. Beal and M. Viroli, "Space–time programming," *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 373, no. 2046, 2015. [Online]. Available: http://rsta.royalsocietypublishing.org/content/373/2046/20140220

[22] F. Damiani, M. Viroli, D. Pianini, and J. Beal, "Code mobility meets self-organisation: A higher-order calculus of computational fields," in *Formal Techniques for Distributed Objects, Components, and Systems*, ser. Lecture Notes in Computer Science, S. Graf and M. Viswanathan, Eds. Springer International Publishing, 2015, vol. 9039, pp. 113–128.

[23] J. Beal and M. Viroli, "Building blocks for aggregate programming of self-organising applications," in *Eighth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, SASOW 2014, London, United Kingdom, September 8-12, 2014*, 2014, pp. 8–13. [Online]. Available: http://dx.doi.org/10.1109/SASOW.2014.6