Programming Abstractions for Augmented Worlds

Angelo Croatti
DISI, University of Bologna
Via Sacchi, 3 – Cesena, Italy
a.croatti@unibo.it

Alessandro Ricci
DISI, University of Bologna
Via Sacchi, 3 – Cesena, Italy
a.ricci@unibo.it

ABSTRACT

The impressive development of technologies is reducing the gulf between the physical and the digital matter, reality and virtuality. In the short future, the design and development of *augmented worlds* – as software systems extending the physical space and environment with computational functionalities and an augmented reality-based appearance – could become an important aspect of programming, calling for novel programming abstractions and techniques. In this paper we introduce this vision, discussing *mirror worlds* as augmented worlds programmed in terms of agent-oriented abstractions.

1. INTRODUCTION

In the short future, the design and development of *augmented worlds* could become an important aspect of programming, not only related to specific application domains—such as the ones traditionally targeted by augmented/mixed/hyper reality [3, 2, 28, 6].

The notion of *augmentation* that we consider in this paper concerns different aspects, spanning from *augmented/mixed reality* as a primary one, to *human augmentation* – enabled by mobile/wearable technologies – and *environment augmentation* – based on pervasive computing and Internet of Things (IoT).

Mixed reality refers to the merging of real and virtual worlds to produce new environments and visualizations where physical and digital objects co-exist and interact in real time [5]. As defined by P. Milgram and F. Kishino, it is "anywhere between the extrema of the virtuality continuum" [13], that extends from the completely real through to the completely virtual environment with augmented reality (AR) and augmented virtuality ranging between. In recent years, there has been an impressive development of hardware technologies related to mobile and wearable supporting different degrees of AR—the main example is given by *smart glasses*. Conceptually, this kind of devices allows to extend people cognitive capabilities, improving the way in which they perform their tasks [26]. This can be interpreted as a form of *human augmentation*, in a way that recalls the Augmented System conceptual framework introduced by the computer science pioneer Doug Engelbart more than 50 years ago [8]. As remarked in [31], it is not only a matter of an hardware augmentation: the software plays a key role here such that we can talk about *software-based* augmentation.

These technologies can have a huge impact from an application point view, allowing for rethinking the way in which people work, interact and collaborate—escaping from the limits that current mobile devices such as smartphones and tablets enforce. A main one is about requiring users to use their hands and watch 2D screens. Smart-glasses allow to free users' hands and perceive application output while perceiving the physical reality where they are immersed.

The vision of pervasive/ubiquitous computing [29, 22, 11, 18], which is more and more entering in the mainstream with the IoT [10, 33], provides a further and related form of augmentation of the physical world. In this case, the augmentation is given by open ecosystems of connected and invisible computing devices, embedded in physical objects and spread in the physical environment, equipped with different kinds of sensors and actuators. Data generated by these devices is typically collected and managed in the cloud, and accessed by mobile/cloud applications. This embedded computing layer *augments* the functionalities of the physical things that people use everyday and, again, are going to have a strong impact on how people work, interact and collaborate [16]. This view about environment augmentation strongly recalls the idea of *computer-augmented environments* by P. Wellner et al. [30], in which the digital (cyber) world is merged with the physical one.

In literature, these forms of augmentation are discussed in separated contexts, mainly focusing on solving issues related to the enabling hardware/software technologies and to develop ad hoc systems for specific application domains. In this paper we are interested to consider these augmentations from a *programming perspective*. In particular we see a convergence that can be captured by the idea of *computation-as-augmentation* of the physical reality, and – more specifically – of computer programs designed to be an extension of the physical environment where people live and work in. To capture this viewpoint, in this work we propose the notion of *augmented worlds* as a conceptual framework to start explor-

ing the main concepts, principles and problems underlying these kinds of programs, their design and development. An augmented world is a computer program augmenting the functionalities of some physical environment by means of full-fledge computational objects *located in the space* that users can perceive and interact with by means of proper mobile/wearable devices. These *augmented entities* – being them objects, actors or agents – can model also extensions of existing physical things, so that their state can be affected by their physical counterpart, and, viceversa, this one can be affected by events/actions occurring at the digital level.

In the remainder of this paper, we first provide an overview of main concepts and features related to augmented worlds (Section 3), abstracting from any specific programming paradigm that can be adopted to implement them. Then, we focus on programming, first providing a general overview of the programming issues that concern the adoption of specific paradigms (Section 4) and then discussing *mirror worlds* (Section 5), which provide a concrete agent-oriented programming model to develop augmented worlds.

Mirror worlds have been introduced in literature in the context of smart environments modelled in terms of agents and multi-agent systems [20]. Inspired by them, the idea of augmented worlds represent a generalization aimed at capturing the main principles concerning these new kinds of systems, as well as discussing the main aspects involved by their design and programming.

Besides introducing the idea of augmented worlds, the aim of this paper is also to provide an overview of some main research challenges and open issues (Section 6), eventually defining a first research agenda for future work.

Before describing more extensively the main concepts that characterize augmented worlds, in next section we review the main research work and technologies that are related to the vision proposed here.

2. BACKGROUND

The level of sophistication achieved by the techniques developed in the context of Augmented/Mixed Reality and Mobile Augmented Reality (MAR) research [3, 5, 24, 23] – supported by the availability of more and more powerful hardware (sensors, processors, displays, wearables) – makes it possible today to focus on the design of augmented worlds assuming that the basic enabling functionalities – such as indoor/outdoor *tracking*, to determine the location and orientation of things in the real world – are available, provided by a bottom layer of the software stack. This does not concern only the *location*—which is however an essential aspect in our case, like in location-based applications. It may include also other elements that more generally define the user *context*—in fact, the techniques developed in context-aware computing [7] are another enabling brick of the vision.

The maturity of these technologies is witnessed by the products and solutions that are entering into the mainstream. Among the others, a main recent one is Microsoft HoloLens [1]—in which we find many points of the augmented world vision. By exploiting an holographic helmet, Microsoft HoloLens generates a multi-dimensional image visible to the user wearing the helmet so that he or she perceives holographic objects in the physical world. Holographic objects are similar to GUI objects whose canvas is the real world—they can be pinned, or anchored, to physical locations chosen by the user, moved according to their own rules, or remain in a specific location within user's field of view regardless of where the user is or in which direction she/he is looking. Beside the many similarities, the kind of augmentation provided in augmented worlds is conceptually different. In particular, holographic objects – as far as authors' understanding from the information currently available about HoloLens [1] – are meaningful only if there is a user who (generates and) perceives them. Conversely, an augmented world has an *objective existence* which does not depend on the users that are located inside: it is first of all a computational augmentation of the environment, which can be then perceived by users with proper devices.

Augmented worlds are typically multi-user systems—a main goal is to ease the development of applications supporting *collaborative* hands-free human activities. From this point view, the augmented worlds vision shares many similarities with online multi-user distributed collaborative environments such as Croquet [25]. The main difference is that, instead of being purely virtual, augmented worlds are deployed in the physical world.

The kind of augmentation typically provided by AR/MAR system is information-oriented, i.e., the objective is to overlay on the physical reality a set of 2D/3D objects that provide some information content about that physical reality, that can be perceived by proper AR browsers [12]. An important exception is given by MAR *games* [27], which have many points in common with the augmented world view. In fact, they typically create multi-user immersive environments blended with the physical reality composed not only by informational objects but computational elements with a behavior—e.g., game bots and characters. In this paper we argue that the usefulness of this kind of extended augmentation goes beyond games, making the investigation of general purpose programming frameworks and abstractions to support them a relevant research topic.

Finally, the vision proposed in this paper is strongly related to any research work that investigates computing systems in which the *physical space* become a first-order concept of the computation layer, beyond the perspectives already developed in distributed and mobile computing. To authors' knowledge, in literature this view has been developed so far by *spatial computing* [32]—whose perspective, however, is quite different from the one discussed in this paper. In fact, spatial computing systems are typically based on very large set of mobile computing devices locally-connected and location-aware that support the execution of distributed computations in which the spatial configuration of the elements has a primary role.

3. AUGMENTED WORLDS – CONCEPTS

In this section we identify and discuss the main features that characterize augmented worlds. These features are mostly independent from the specific programming paradigms that can be adopted to implement them – for this reason we will try to abstract as much as possible from them.

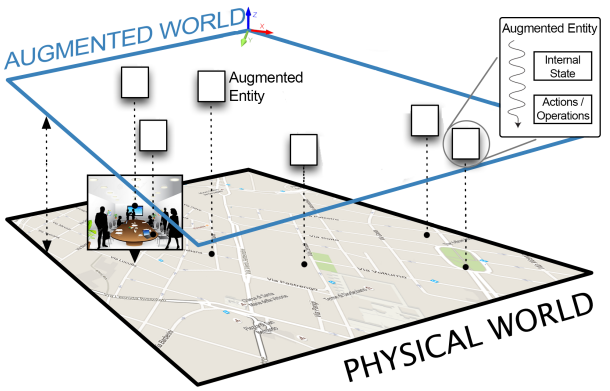

Figure 1: A simple representation of an Augmented World, focusing on spatial coupling aspect of augmented entities.

3.1 Spatial Coupling

The main distinguishing feature that characterizes augmented worlds is to be composed by computational objects that are situated in some specific location of the physical reality (see Figure 1). In the following we will refer to these computational objects as *augmented entities*, abstracting from the specific programming paradigm that be used to implement them – they could be objects in OOP, actors in actor programs, etc.

An augmented entity is instantiated at runtime by specifying also its location, that could be either directly a geographical position or indirectly the reference to a physical object living in the physical reality. The location can be specified with respect to a local system of reference defined by the augmented world. An important point here is that augmented entities are meant to model not just data items – like in the case of POIs (Point-of-Interests) as found in MAR applications – but any full-fledged computational entity, eventually encapsulating a state and a behaviour. This is the basic fundamental feature which is useful in particular to build applications where the augmentation is not just an information layer about physical things, but more complex services or functionalities involving processing and interaction.

It is worth clarifying that an augmented entity is bound to some physical location not (necessarily) because it is running on some hardware device physically situated at *that* location—like in the case of e.g., spatial computing [32]. The code, data and runtime state of augmented entities are meant to be managed by one or multiple computing server nodes hosting augmented worlds execution, possibly in the cloud.

Besides the location, an augmented entity can have a *geometry* modeling the extension of the object in the physical space, like in classic AR application. This geometry can be described e.g., as a set of 3D points defined with respect to a local system of reference.

Both the location and the geometry are part of the computational state of the entity, and can change at runtime then by virtue of the computation and interaction occurring inside the augmented world.

3.2 Discovery and Observability

The spatiality of augmented entities can be exploited to enrich the ways in which these computational objects are discovered, observed and more generally how they interact with each other—including interaction with users.

About discovery, inside an augmented world (as a system in execution), the reference to an augmented entity could be retrieved by *lookup* functions – provided by the augmented world runtime – specifying the region of the world to be queried, among the possible filters. This is essentially a pull-based approach to discovery.

Besides, a push-based/event-driven modality is useful as well, that is: retrieving the reference to an augmented entity as soon as it becomes *observable*, given its position with respect to the position of the *observing* entity. In order to support this modality, the augmented entity abstraction can be further characterized by two observation-related properties: an *observability neighborhood* and an *observation neighborhood*, defining a spatial-based constraint on the set of entities that – respectively – can observe/perceive the augmented entity and can be observed/perceived by the augmented entity. The simplest kind of spatial neighborhood can be defined in terms of distances between the position where the entities are located. In this case, these properties can be simply referred as an *observability radius* and *observation radius* (see Figure 2, left).

As a simple example about the usefulness of these properties, an augmented entity representing a message can be left in a specific point in a city and can be observed (read) only by entities that are in message's proximity (e.g., the ones representing human users immersed in the augmented world). As another example, a counter entity left in a specific geographical location with the purpose to count how much entities pass near the counter, may be defined to perceive all entities in a very short observation radius, but the observability radius can be set to zero, so other entities does not necessarily perceive the counter.

Also the properties related to observability can change at runtime, depending on the functionality of the augmented entity, its computing behaviour and interaction within the world.

3.3 User Modeling and Interaction

User interaction is a main aspect of augmented worlds, whose primary objective is to model applications where human users – and robots, eventually – are engaged in some kind activity in some physical environment.

An augmented world is typically a multi-user application, where multiple users perceive, share and interact within the same augmented entities. An effective way to model a user in an augmented world is to associate her/him with an augmented entity – as a kind of *augmented body* – with features that are useful for controlling user input/output and a computational state reifying the dynamic information about the user (state) that can be relevant at the application level. By

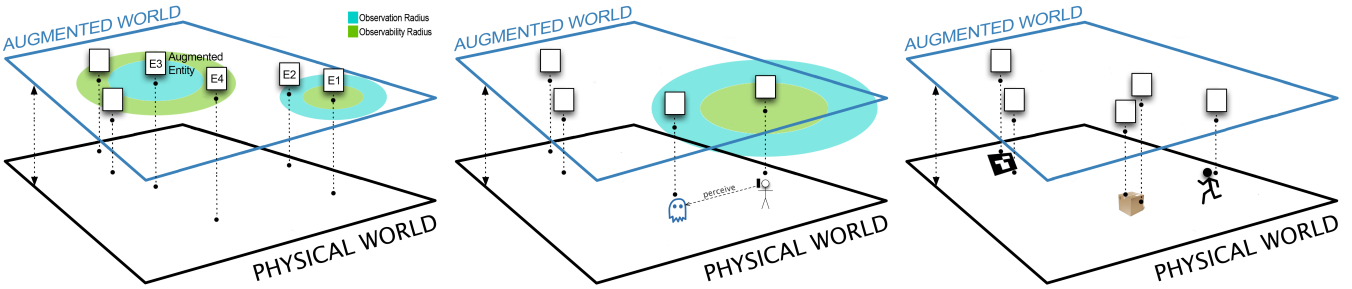

Figure 2: (left) In this time instant, the entity E1 can perceive E2 but cannot be observed by other entities (no entities within its observability radius). Viceversa, E3 can perceive E4 and can be observed by all others entities within its observability radius. (center) An human wearing glasses can perceive ghost entity through its associated assistant entity. (right) Physical Embedding and Coupling issue representation.

exploiting the observation-related features, it is possible to determine which entities of the world the user is perceiving and their observable state (see Figure 2, center). This state can include also the geometry of the entity, which can be eventually exploited for constructing the view perceived by the user.

Besides, a key aspect of this modeling is that it allows for programming augmented entities that perceive the presence of the users, by exploiting the same mechanisms. This feature can be useful in particular to simplify the design of smart environments that react to events related to user state and behavior, reified in the corresponding augmented body.

3.4 Physical Embedding

The notion of augmentation in augmented worlds eventually means also the extension of the functionalities of existing physical objects (see Figure 2, right). Such an extension can be useful at two different (but related) levels.

A first one is for human users interacting with the physical objects, extending the object affordances by means of virtual interfaces to control or inspect the state of the physical object.

A second one is for enriching the physical object functionalities, exploiting the computing capabilities given by the augmented layer (either embedded or not in the object itself). For instance, an alarm clock on the bedside table – exploiting the associated augmented entity – can provide to a user the functionality to seamless schedule wake-up times, considering morning appointments on calendar or – in the short future – considering the monitored sleep status (by another augmented device) and adjusting wake-up timing consequently.

This kind of augmentation requires in general some kind of *coupling* between the physical objects and the corresponding augmented entities providing their extension, such that changes to the physical state of the objects are tracked and reified in the augmented level. In other words, the augmented level always needs to be informed about changes into the physical world. When this is not possible, due to e.g. a lack of connectivity, the augmented world's infrastructure has to update a kind of meta-information for each information/data related to physical world (like a "degree of freshness") to infer if and when a specific information/data could be no longer aligned to the real state.

4. TOWARDS A PROGRAMMING MODEL

After sketching the main characteristics of augmented worlds, we consider now the issue of how to implement them, what kind of programming abstractions can be adopted.

OOP provides an effective approach to model the basic structural aspects of augmented worlds. Augmented entities can be directly mapped into simple objects – encapsulating augmented entity state and behaviour – possibly exploiting interfaces and classes to define the type and hierarchies of augmented entities. An augmented object would have basic operations to manage aspects such as the position inside the augmented world. An augmented world in this case can be modelled as the container of the augmented objects, providing services for their management. The observer pattern could be used in this case to model the observability features discussed in previous section.

This OOP modeling however is not effective to capture concurrency and asynchronous interactions, which are main fundamental aspects of this kind of systems. Augmented worlds are inherently concurrent systems. From a control point of view, augmented entities are independent computing entities, whose computations can be triggered asynchronously by virtue of different kind of events—user(s) actions and other entities requests. At the logical level, they are distributed, given their spatial coupling.

These aspects can effectively captured by concurrency model integrating objects with concurrency, such actors and actor-like entities such as active/concurrent objects. This choice allows for strengthening the level of encapsulation, devising augmented worlds as collections of *autonomous* entities encapsulating a state, a behaviour and the control of the behaviour. Modelling augmented entities as actors means using direct asynchronous message passing to model every kind of interaction occurring inside an augmented world. In this case, the observation-related features can be implemented as a framework on top, by means of, e.g., pre-defined actors providing functionalities related to lookup/discovery and spatial-driven observation. Alternatively, these features

can be embedded in the actor framework/language, extending the basic model.

The observation-related features can be directly captured instead by adopting an *agent-oriented* modelling. In that case, an augmented world can be modeled in terms of autonomous agents situated into a virtual environment making the bridge between the physical and augmented layer. The augmented entities would be the basic bricks composing such environment, which can be observed and manipulated by the agents by means of the environment interface (i.e. the set of actions/percepts).

In next section we go deeper in this modeling, by discussing a first example of agent-oriented augmented world programming based on *mirror worlds*.

5. THE CASE OF MIRROR WORLDS

Mirror words (MW) have been introduced in [20] as an agent-oriented approach to conceive the design of future smart environments, integrating in the same unifying model perspectives and visions from Distributed Systems [9], Ambient Intelligence (AmI) and Augmented Reality. In the MW vision, smart spaces are modelled as digital cities shaped in terms of the physical world to which they are coupled, inhabited by open societies and organisations of *software agents* playing the role of the inhabitants of those cities. *Mirroring* is given by the fact that physical things, which can be perceived and acted upon by humans in the physical world, have a digital counterpart (or augmentation, extension) in the mirror, so that they can be observed and acted upon by agents. Viceversa, an entity in the MW that can be perceived and acted upon by software agents, may have a physical appearance (or extension) in the physical world—e.g., augmenting it, in terms of AR, so that it can be observed and acted upon by humans.

Mirror worlds are based on the A&A (Agents and Artifacts) [14] meta-model, introduced in agent-oriented software engineering to exploit also the agent environment as a first-order abstraction aside to agents to model and design multi-agent systems. In particular, A&A introduces *artifacts* as first-class abstractions to model and design the application environments where agents are logically situated. An artifact can be used to model and design any kind of (non-autonomous) resources and tools used and possibly shared by agents to do their job [21]. Artifacts have an *observable state* that agents can perceive and a set of *operations*, that agents can request as *actions* to affect the world. The observable state is represented by a set of *observable properties*, whose value can change dynamically depending on the actions executed on the artifact. Like objects in OOP, artifacts can have also a hidden state, not accessible to agents. The concept of observable state is provided to support event-driven forms of interaction between agents and artifacts, so that an agent observing an artifact is asynchronously notified with an event each time the state of the artifact is updated. Artifacts are collected in *workspaces*, which represent logical containers defining the topology of the multi-agent system, which may be distributed over the network.

The interactions among agents and artifacts are fully asynchronous. Actions on artifacts executed by agents – which encapsulate their own logical thread of control, like actors – are executed by separate threads of control with respect to the agent one. Operations are executed *transactionally*, so the execution of multiple actions concurrently on the same artifact is safe [21].

5.1 Mirror Worlds as Augmented Worlds

Given the augment worlds conceptual framework introduced in this paper, a mirror world can be described as an agent-oriented augmented world, implementing some of the principles that have been discussed in Section 3.

Artifacts and workspaces are used in mirror worlds to model the bridge between the augmented world layer and the physical reality layer. In particular, a MW is modelled in term of a set of *mirror workspaces*, which extend the notion of workspace defined in A&A with a *map* specifying which part of the physical world is coupled by the MW. It could be a part a city, a building, a room. The map defines a local system of reference to locate *mirror artifacts* inside. Mirror artifacts are simply artifacts anchored to some specific location inside the physical world, as defined by the map. Such location could be either a geo-location, or some trackable physical marker/object. Thus, mirror artifacts realize the spatial coupling defined in Section 3.

About observability, in a MW an agent can perceive and observe a mirror artifact in two basic ways. One is exactly the same as for normal artifacts, that is explicitly *focusing* on the artifact, given its identifier [21]—focusing is like a subscribe action in publish/subscribe systems. The second one instead is peculiar to mirror workspace and is the core feature of agents living in mirror workspaces, that is: perceiving an artifact depending on its position, without the need of explicit subscription (focusing). To that purpose, an agent joining a mirror workspace can create a *body* artifact, which is a built-in mirror artifact useful to situate the agent in a specific location of the workspace. We call *mirror agent* an agent with a body in a mirror workspace. Observability is ruled by two parameters – as defined in Section 3 – the observability radius and the observation radius. The observability radius is defined for each mirror artifact and defines the maximum distance within which an agent (body) must be located in order to perceive the artifact. The observation radius instead is defined for each agent body and defines the maximum distance within which a mirror artifact must be located in order to be perceived by an agent. Thus, a mirror artifact X located at the position X_{pos}, with an observability radius X_r is observable by a mirror agent with a body B, located in B_{pos}, with an observation radius B_R, iff $d <= X_r$ and $d <= B_R$, being d the distance between X_{pos} and B_{pos}. Both parameters can be changed at runtime.

The user interface in MW is currently realized by user assistant mirror agents with a body coupled to the physical location of the human user, by means of a smart device—glass, phone, whatever. Such agents perceive mirror artifacts in the nearby of the user location, their observable state, and can select and represent them on the smart-glass worn by the user. The representation can range from simple messages and cues to full-fledged augmented reality rendering, eventually superimposing virtual 3D objects on the image

of the physical reality.

About the physical embedding discussed in Section 3, mirror artifacts can be either completely virtual, or coupled to some object of the physical reality. In the first case, the geo-position inside the mirror (and the physical environment) is specified when instantiating the artifact, and it can be updated then by operations provided by the artifact. In the second case, at the infrastructure level, the state and position of the mirror artifact is kept *synchronized* to the state and position of the physical object by the MW runtime, by means of suitable sensors/embedded devices.

5.2 Programming Mirror Worlds

A first implementation of Mirror Worlds has been developed on top of the JaCaMo agent framework [19], which directly supports the A&A meta-model. Agents are programmed using the Jason agent programming language [4], which is a practical extension and implementation of AgentSpeak(L) [17]; Artifact-based environments are programmed using the CArtAgO framework [21], which provides a Java API for that purpose.

The MW API are currently a layer on top of JaCaMo, including:

- `MirrorArtifact` artifact template, extending the `Artifact` CArtAgO base class and representing the basic template to be further extended and specialized to implement specific kinds of mirror artifacts. The usage interface of this artifact includes:
  - a `pos` observable property, containing the current location in the mirror workspace of the artifact;
  - `observabilityRadius` observable property, storing the current observability radius of the artifact;
  - specific operations (`setPos`, `setObservability Radius`) for updating the position and the observability radius.

  The usage interface of agent bodies includes also an `observationRadius` observable property, storing the current observation radius of the agent, and the related operation `setObservationRadius` for updating such radius.
- a new set of actions to be used by agents for creating mirror workspaces and mirror artifacts inside, including agent bodies.
- some *utility* artifacts are available providing functionalities useful for agents working in the mirror. An example is given by the `GeoTool`, which provides functionalities for converting the coordinates and computing distances.

In the remainder of the section we give an overview of MW programming in JaCaMo by considering a simple example of mirror world, a kind of *hello, world*. An overview of the main elements of the example is shown in Figure 3. The mirror world is composed by a single mirror workspace (called `mirror-example`) mapped onto a city zone in the

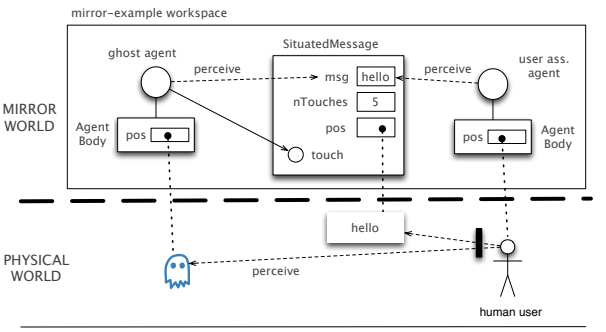

Figure 3: Main elements included in the example discussed in Section 5: a mobile user walking along the streets and the corresponding user assistant agent, with a body located at the position detected by the GPS. A situated message, modelled as a `SituatedMessage` mirror artifact. A ghost, as a mirror agent autonomously walking through the streets.

center of a city (Cesena, in this case, in Italy). The mirror workspace is dynamically populated of mirror artifacts representing messages situated in some specific point of the city (template `SituatedMessage`). Besides keeping track of a message, these artifacts embed a counter and a `touch` operation to increment it. Mobile human users walk around the streets along with their user assistant agents, running on their mobile device (e.g, the smartphone or directly the smart glasses). As soon as user assistant agents perceive a situated message, they display it on the smart glasses worn by the users. In the MW there are also some `ghost` mirror agents that are moving around autonomously along some streets of the city, perceiving and interacting with the situated messages as well. They react to situated messages perceived while walking, eventually executing a `touch` action on the mirror artifact encountered. Besides, if/when a ghost agent reaches a human user, it *hugs* her/him, which is physically perceived by a trembling of the smartphone. This occurs by executing a `tremble` action on the artifact modeling the user mobile device.

The example includes also a *control room* (see Figure 4), which is a remote desktop application which allows to track the real-time state of the running mirror world, showing the position of mirror agents (red circles) – actually the body of mirror agents – and the position of mirror artifacts, i.e. the situated messages in the example.

This simple example contains most of the main ingredients of an augmented/mirror world. The situated messages represent stateful augmented entities, with a simple behaviour; Such augmented entities are shared, perceived and manipulated by both the human users (indirectly through the user assistant agents) and the other autonomous agents living in the mirror—i.e., the ghosts. Through the mirror, such agents can have an effect also on the physical world (trembling of the smartphones). An aspect which is quite overlooked in the example is the visual representation of augmented objects, which is currently fairly simple (just messages)—more sophisticated AR-based views are planned

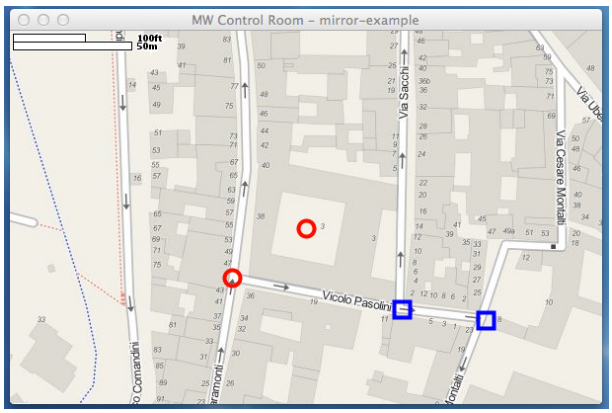

Figure 4: The map visualised by the control room, showing the position of mirror agents (red circles) – that is, the body of mirror agents – and the position of mirror artifacts, i.e. situated messages in the example.

in the future.

In order to have a concrete taste of MW programming, in the following we show some details about how the mirror agents and artifacts are programmed—the full code is available in [15], along with the experimental JaCaMo distribution supporting mirror worlds.

5.2.1 *Defining Mirror Artifacts*

The situated messages of the example are implemented by the the `SituatedMessage` mirror artifact—Figure 5 shows the implementation of the Java class representing the artifact template. Artifacts in CArtAgO can be defined as classes extending the `Artifact` base class. Methods annotated with `@OPERATION` define the operations available to agents. Observable properties are managed by means of predefined primitives (e.g. `defineObsProperty`, `getObsProperty`, ...)—implemented as protected methods of the base class. Operation execution is atomic, so – similarly to monitors – only one operation at a time can be running inside an artifact. Changes to the observable properties are made observable to agents only when an operation has completed. Further details about the artifact model are described in [21].

A mirror artifact can be defined by extending the base `MirrorArtifact` class—which is an extension itself of the CArtAgO `Artifact` class. `SituatedMessage` has two observable properties (besides the ones inherited by `MirrorArtifact`), `msg` and `nTouches`, storing the content of the message and the counter keeping track of the number of times that the message has been *touched*. The `touch` operation allows to increment the counter.

5.2.2 *Implementing Agents in the Mirror*

Agents in the mirror are normal Jason agents, with more actions available—given by the new artifacts introduced by the MW framework. In particular, specific actions are available for creating mirror workspaces and instantiating mirror artifacts. As an example, Figure 6 shows the code of

```
1  public class SituatedMessage extends MirrorArtifact {
2
3      public void init(String msg){
4          super.init(msg);
5          defineObsProperty("msg",msg);
6          defineObsProperty("nTouches",0);
7      }
8
9      @OPERATION void touch(){
10         updateObsProperty("nTouches",
11               getObsProperty("nTouches").intValue()+1);
12     }
13 }
```

Figure 5: Source Code of the mirror artifact representing a situated message.

```
1  /* initial beliefs */
2
3  /* the center of the mirror -- latitude/longitude */
4  poi("isi_cortile", 44.13983, 12.24289).
5
6  /* the point of interests, where to put the messages */
7  poi("pasolini_montalti",44.13948, 12.24384).
8  poi("sacchi_pasolini",44.13952, 12.24340).
9
10 /* initial goal*/
11 !setupMW.
12
13 /* the plans */
14
15 +!setupMW
16   <- ?poi("isi_cortile",Lat,Long);
17      createMirrorWorkspace("mirror-example",Lat,Long);
18      joinWorkspace("mirror-example");
19      /* create an aux artifact to help coordinate conversion */
20      makeArtifact("geotool","GeoTool",[Lat,Long]);
21      /* create the situated messages */
22      !create_messages;
23      println("MW ready.").
24
25 /* to create the situated message mirror artifacts */
26 +!create_messages
27   <- ?poi("pasolini_montalti",Lat,Lon);
28      toCityPoint(Lat,Lon,Loc);
29      createMirrorArtifactAtPos("a1","SituatedMessage",
30          ["hello #1"],Loc,2.5);
31      ?poi("sacchi_pasolini",Lat2,Lon2);
32      toCityPoint(Lat2,Lon2,Loc2);
33      createMirrorArtifactAtPos("a2","SituatedMessage",
34          ["hello #2"],Loc2,2.5).
```

Figure 6: Code of the `majordomo` agent.

a *majordomo* agent, whose task is to setup the initial environment of the MW of our example. The agent creates a mirror workspace, called `mirror-example` (line 17), and a couple of `SituatedMessage` mirror artifacts (plan at lines 26-34), located at two POIs tagged as `pasolini_montalti` and `sacchi_pasolini` (which are two street intersections on the map). The action `createMirrorArtifactAtPos` makes it possible to instantiate a new mirror artifact, specifying its logical name, the template (the Java class name), its location `pos` and the observability radius (in meter). The utility artifact (`GeoTool` template) used by the agent provides functionalities to manage geographical coordinates, in particular to convert global latitude/longitude coordinates into the local one of the mirror workspace (`toCityPoint` operation).

The first example of mirror agent is given by *user assistant agents*, whose code is shown in Figure 7. The agent first cre-

ates (in its default/local workspace) a `SmartGlassDevice` artifact (line 8), to be used as output device to display messages, by means of the `displayMsg` operation. Then, the agent joins the mirror workspace and creates its body, with observation radius of 10 meters—to this purpose the `createAgentBody` action is used, specifying the observing and observability radii (line 14). The body is bound to a `GPSDeviceDriver` device driver artifact (line 18), previously created (line 16). The device driver implements the coupling between the position detected by the GPS sensor, available on the smartphone of the user. When the human user approaches a point in the physical world where a situated message is located, the user assistant agent perceives the message and reacts by simply displaying it on the glasses (lines 23-25). When (if) the human user moves away from the mirror artifact, the belief about the message is removed and the use assistant agent reacts by displaying a further message (lines 27-29).

Figure 8 shows the code of the ghost mirror agents, autonomously walking through some streets of the mirror world. They have a `walk_around` goal (line 8), and the plan for that goal (line 12) consists in repeatedly doing the same path, whose nodes (a list of point-of-interests) is stored in the `path` belief (line 5). They move by changing the position of their body, by executing a `moveTowards` action available in each mirror artifact—specifying the target point (to define the direction) and the distance to be covered (in meters). The plan for reaching an individual destination of the path (lines 23-29) simply computes the distance from the target (exploiting the `computeDistanceFrom`, provided by the `GeoTool` artifact) and then, if the distance is greater than one meter, it moves the body of 0.5 meter and then goes on reaching, by requesting recursively the subgoal `!reach_dest`; otherwise it completes the plan (the destination has been reached). Ghosts too react to messages perceived while walking (plan at lines 38-41), eventually executing a `touch` action on each message encountered and printing to the console the current number of touches observed on the message. Instead, when a ghost perceives a human (lines 43-46) – by perceiving the body of the user assistant agent – it reacts by making a trembling on the smartphone owned by the human user. `body` is an observable property provided by each agent body artifact, containing the identifier of the user assistant agent which created the body. Trembling happens by executing a `tremble` action on the artifact which the user assistant agent created to enable the physical interaction with the corresponding human user. By convention, in the example, these artifacts are created with the name `user-dev-X`, where `X` is name of the user assistant agent. This convention allows the ghost agent to retrieve the identifier of the artifact dynamically given its logic name, by doing a lookup (line 45).

5.3 Remarks

The example, in spite of its simplicity and of the details about Jason/CArtAgO programming, should provide a first idea about the level of abstraction provided by agent-oriented programming for designing and programming augmented worlds. The main strength is that it allows to model the augmented world in a way which is similar to the real world—even more similar in our opinion than the modeling provided by paradigms such as actors or concurrent objects.

```
1  /* User assistant agent */
2
3  /* goal of the agent */
4  !monitor_and_display_messages.
5
6  +!monitor_and_display_messages
7    <- /* setup the smart glass device */
8       makeArtifact("viewer","SmartGlassDevice",[],Viewer);
9       /* keep track of the device id with a viewer belief */
10      +viewer(Viewer);
11      /* join the mirror workspace */
12      joinWorkspace("mirror-example");
13      /* create the agent body */
14      createAgentBody(1000,10,Body);
15      /* create the artifact used as MW coupling device */
16      makeArtifact("driver","GPSDeviceDriver",Dev);
17      /* bind the body to the device */
18      bindTo(Body)[artifact_id(Dev)];
19      println("ready.").
20
21 /* plans reacting to situated messages perceived in the mirror worlds */
22
23 +msg(M) : viewer(Dev)
24   <- .concat("new message perceived: ",M,Msg);
25      displayMsg(100,50,Msg)[artifact_id(Dev)].
26
27 -msg(M) : viewer(Dev)
28   <- .concat("message ",M," no more perceived. ",Msg);
29      displayMsg(100,50,Msg)[artifact_id(Dev)].
```

Figure 7: Code of the user-assistant agents.

```
1  /* ghost agent initial beliefs */
2
3  start_pos("pasolini_chiaramonti").
4  /* path of the walk - 2 steps*/
5  path(["sacchi_pasolini","pasolini_montalti"]).
6
7  /* initial goal */
8  !walk_around.
9
10 /* plans */
11
12 +!walk_around <- !setup; !moving.
13
14 +!moving <- ?path(P); !make_a_trip(P); !moving.
15
16 +!make_a_trip([POI|Rest])
17   <- ?poi(POI,Lat,Lon);
18      !reach_dest(Lat,Lon);
19      !make_a_trip(Rest).
20 +!make_a_trip([])
21   <- ?start_pos(Start); ?poi(Start,Lat,Lon); !reach_dest(Lat,Lon).
22
23 +!reach_dest(Lat,Lon) : myBody(B)
24   <- toCityPoint(Lat,Lon,Target);
25      computeDistanceFrom(Target,Dist)[artifact_id(B)];
26      if (Dist > 1){
27        moveTowards(Target,0.5)[artifact_id(B)];
28        .wait(50);
29        !reach_dest(Lat,Lon)}.
30
31 +!setup
32   <- joinWorkspace("mirror-example",Mirror);
33      lookupArtifact("geotool",Tool); focus(Tool);
34      ?start_pos(Point); ?poi(Point,Lat,Lon); toCityPoint(Lat,Lon,P);
35      createAgentBodyAtPos(P,1000,10,Body);
36      +myBody(Body); .my_name(Me); +me(Me).
37
38 +msg(M) [artifact_id(Id)]
39   <- touch [artifact_id(Id)];
40      ?nTouches(C)[artifact_id(Id)];
41      println("new message perceived: ",M," - touch count: ",C).
42
43 +body(Who) : me(Me) & Who \== Me
44   <- .concat("user-dev-",Who,Dev);
45      lookupArtifact(Dev,DevId);
46      tremble [artifact_id(DevId)].
```

Figure 8: Code of ghost agents.

In particular, both in the real world and the augmented worlds, a main role is played by indirect interactions (vs. direct message passing) based on the (asynchronous) observation of events occurring in the environment. This is directly captured by the agent/environment abstractions.

6. CHALLENGES AND FUTURE WORK

The development of augmented worlds relies on the availability of enabling technologies that deal with issues and challenges of different kinds. An example is given by tracking and registration, which are a main concern of the AR and MAR layer [3, 5].

Besides the challenges in the enabling layers, there are further issues that specifically concern the augmented worlds model.

A main general one is related to the level of real-time coupling and synchronization between the computational augmented layer and the physical layer. This coupling/synchronization is critical from users' perspective, since it impacts on what users perceive of an augmented world, and then how they reason about it and act consequentially. Being a multi-user system, two users must perceive the same observable state of the shared augmented entities. If a part of the augmented world is temporarily disconnected – because of, e.g., some network transient failure – users must be able to realize this.

These aspects are challenging in particular because – like in distributed systems in general – it is not feasible in an augmented world to assume a single clock defining a centralized notion of time. Conversely, it is fair to assume that each augmented entity has its own local clock and the events generated inside an augmented world can be only *partially* ordered. In spite of the distribution, causal consistency must be guaranteed, in particular related to chains of events that span from the physical to the digital layer and viceversa. That is, if an augmented entity produces a *sequence* of two events concerning the change of its observable state, the same sequence must be observed by different human users immersed in the augmented world.

Similar challenges are found in online multi-user distributed collaborative systems. As a main example, Croquet is based on TeaTime [25], a scalable real-time multi-user architecture which manages the communication among objects, and their synchronization. The spatial coupling and physical embedding properties of augmented worlds introduce further elements and complexities, that are not fully captured by strategies adopted in purely virtual systems.

Finally, the aim of this paper was to introduce the vision about augmented worlds, along with a first conceptual framework discussing some main features of their programming abstractions. Clearly, a more rigorous and comprehensive approach is needed to tackle the development and engineering of non-naive augmented worlds, and, more generally, to achieve a deeper understanding of the *computation-as-augmentation* view. This understanding includes, e.g., investigating if and how spatial coupling impacts on system modularity, compositionality, extensibility. To this purpose, the definition of formal models capturing the main aspects and properties of this kind of programs appears an important future work. Another important investigation concerns the design of proper tools supporting the development/debugging/profiling of augmented worlds. These tools must provide specific features to deal with the characteristics described in Section 3. To this purpose, the design of proper real-time *simulators* – allowing to run an augmented world like, e.g., a first-person perspective video-game – appears an interesting solution to explore.

7. CONCLUSION

The fruitful integration of enabling technologies concerning augmented reality, mobile/wearable computing and pervasive computing makes it possible to envision a new generation of software systems in which computation and programming can be exploited to shape various forms of augmentation of the physical reality.

Augmented worlds – introduced in this paper – are programs that are meant to extend the physical world by means of full-fledge computational entities logically situated in some physical location, possibly enriching the functionalities of existing physical objects. Mirror words [20, 19] provide a concrete agent-oriented programming model for building augmented worlds—based on the A&A conceptual model and the JaCaMo platform.

The main contribution of this work is to lay down the first basic bricks about the augmented worlds vision, and to trigger further research and practical investigations, including the engineering of real-world and robust applications based on these ideas.

8. REFERENCES

[1] Microsoft HoloLens, Official web site. https://www.microsoft.com/microsoft-hololens.

[2] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, and B. MacIntyre. Recent advances in augmented reality. *Computer Graphics and Applications, IEEE*, 21(6):34–47, 2001.

[3] R. T. Azuma et al. A survey of augmented reality. *Presence*, 6(4):355–385, 1997.

[4] R. H. Bordini, J. F. Hübner, and M. Wooldrige. *Programming Multi-Agent Systems in AgentSpeak using Jason*. Wiley Series in Agent Technology. John Wiley & Sons, 2007.

[5] E. Costanza, A. Kunz, and M. Fjeld. Human machine interaction. chapter Mixed Reality: A Survey, pages 47–68. Springer-Verlag, Berlin, Heidelberg, 2009.

[6] K. Curran, D. McFadden, and R. Devlin. The role of augmented reality within ambient intelligence. *Int. Journal of Ambient Computing and Intelligence*, 3(2):16–33, 2011.

[7] A. K. Dey. Understanding and using context. *Personal and ubiquitous computing*, 5(1):4–7, 2001.

[8] D. C. Engelbart and W. K. English. A research center for augmenting human intellect. In *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, AFIPS '68 (Fall, part I), pages 395–410, New York, NY, USA, 1968. ACM.

[9] D. H. Gelernter. *Mirror Worlds: or the Day Software Puts the Universe in a Shoebox...How It Will Happen*

and What It Will Mean. Oxford, 1992.

[10] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Gener. Comput. Syst.*, 29(7):1645–1660, 2013.

[11] S. Kurkovsky. Pervasive computing: Past, present and future. *5th IEEE International Conference on Information and Communications Technology (ICICT)*, 2007.

[12] T. Langlotz, T. Nguyen, D. Schmalstieg, and R. Grasset. Next-generation augmented reality browsers: Rich, seamless, and adaptive. *Proceedings of the IEEE*, 102(2):155–169, Feb 2014.

[13] P. Milgram and F. Kishino. A taxonomy of mixed reality visual displays. *IEICE Trans. Information Systems*, E77-D(12):1321–1329, Dec. 1994.

[14] A. Omicini, A. Ricci, and M. Viroli. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3):432–456, 2008.

[15] Pervasive Software Lab – DISI, University of Bologna. JacaMo-MW – mirror worlds in JaCaMo. https://bitbucket.org/pslabteam/mirrorworlds, 2015.

[16] R. Poovendran. Cyber-physical systems: close encounters between two parallel worlds. *Proceedings of the IEEE*, 98(8), 2010.

[17] A. S. Rao. Agentspeak (l): Bdi agents speak out in a logical computable language. In *Agents Breaking Away*, pages 42–55. Springer, 1996.

[18] S. Reeves. Envisioning ubiquitous computing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 1573–1582. ACM, 2012.

[19] A. Ricci, A. Croatti, P. Brunetti, and M. Viroli. Programming Mirror-Worlds: an Agent-Oriented Programming Perspective. In *Engineering Multi-Agent Systems Third International Workshop, EMAS 2015, Revised Selected Papers*, LNCS. Springer, 2015. To Appear.

[20] A. Ricci, M. Piunti, L. Tummolini, and C. Castelfranchi. The mirror world: Preparing for mixed-reality living. *IEEE Pervasive Computing*, 14(2):60–63, 2015.

[21] A. Ricci, M. Piunti, and M. Viroli. Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, 23(2):158–192, Sept. 2011.

[22] M. Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal Communications*, 8:10–17, 2001.

[23] D. Schmalstieg, T. Langlotz, and M. Billinghurst. Augmented reality 2.0. In G. Brunnett, S. Coquillart, and G. Welch, editors, *Virtual Realities*, pages 13–37. Springer Vienna, 2011.

[24] D. Schmalstieg and G. Reitmayr. The world as a user interface: Augmented reality for ubiquitous computing. In G. Gartner, W. Cartwright, and M. Peterson, editors, *Location Based Services and TeleCartography*, Lecture Notes in Geoinformation and Cartography, pages 369–391. Springer Berlin Heidelberg, 2007.

[25] D. A. Smith, A. Kay, A. Raab, and D. P. Reed. Croquet-a collaboration system architecture. In *Creating, Connecting and Collaborating Through Computing, 2003. C5 2003. Proceedings. First Conference on*, pages 2–9. IEEE, 2003.

[26] T. Starner. Project Glass: An Extension of the Self. *Pervasive Computing, IEEE*, 12(2):14–16, April 2013.

[27] B. H. Thomas. A survey of visual, mixed, and augmented reality gaming. *Comput. Entertain.*, 10(3):3:1–3:33, Dec. 2012.

[28] J. Tiffin and N. Terashima. *HyperReality: Paradigm for the Third Millenium*. Routledge, 2001.

[29] M. Weiser. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(3):3–11, July 1999.

[30] P. Wellner, W. Mackay, and R. Gold. Computer-augmented environments: back to the real world. *Communications of the ACM*, 36(7), 1993.

[31] C. Xia and P. Maes. The design of artifacts for augmenting intellect. In *Proceedings of the 4th Augmented Human International Conference*, pages 154–161. ACM, 2013.

[32] F. Zambonelli and M. Mamei. Spatial computing: An emerging paradigm for autonomic computing and communication. In M. Smirnov, editor, *Autonomic Communication*, volume 3457 of *Lecture Notes in Computer Science*, pages 44–57. Springer Berlin Heidelberg, 2005.

[33] D. Zhang, L. T. Yang, and H. Huang. Searching in internet of things: Vision and challenges. *in Proc. IEEE 9th Int. Symp. Parallel Distrib. Process. Appl. (ISPA)*, 2011.