A Formal Programming Framework for Digital Avatars[⋆]

Alejandro Pérez-Vereda, Carlos Canal, and Ernesto Pimentel

University of Malaga, Spain
apvereda@uma.es, canal@lcc.uma.es, ernesto@lcc.uma.es

Abstract. In the current IoT era, the number of smart things to interact with is raising everyday. However, each one of them precises a manual and specific configuration. In a more people-friendly scenario, smart things should adapt automatically to the preferences of their users. In this field, we have participated in the design of People as a Service, a mobile computing reference architecture which endows the smartphone with the capability of inferring and sharing a virtual profile of its owner. Currently, we are developing Digital Avatars, a framework for programming interactions between smartphones and other devices. This way, the smartphone becomes a personalized and seamless interface between people and their IoT environment, configuring the smart things with information from the virtual profile. In this work, we present a formalization of Digital Avatars by means of a Linda-based system with multiple shared tuple spaces.

Keywords: Digital Avatars, People as a Service, PeaaS, Linda, Shared Tuple Spaces.

1 Introduction

The Internet of Things (IoT) is built over a layer of connected devices and sensors that offer specific interfaces to access the information they collect and also to configure how they work, e.g. the frequency to pick up the data or how to format them [9]. Recent research in the IoT field has promoted the development of devices and sensors which are more configurable and provide easier interfaces. They are known as smart things [11]. However, smart things still require a lot of manual configuration, and this problem becomes more challenging the bigger the number of devices we daily interact with. In a desirable scenario, the technology should work for the people and not the other way around. Every smart thing should adapt to the needs of the people seamlessly and in an automatic way, reducing the need for interaction with the users to the minimum.

Considering the pervasive presence of smartphones, the authors of this paper have participated in the design of a mobile computing reference architecture

⋆ This work has been funded by the Spanish Government under grants PGC2018-094905-B-I00 and TIN2015-67083-R (MINECO/FEDER).

called People as a Service (PeaaS) [10]. This architecture promotes the use of smartphones to learn about their users, creating and storing virtual profiles with their preferences and context information. These profiles are then offered as a service to third parties in a secure manner. This way, smartphones become seamless and automatic interfaces that negotiate their owner's preferences, adapting and configuring the smart things in their surroundings.

For that purpose, the required interactions are not just simple data transfers, but we need mechanisms that allow to configure smart things, and also to complete virtual profiles with context knowledge obtained from these interactions. The more complete virtual profiles are, the better may the technology adapt to the people. With this goal in mind, we are developing Digital Avatars, a dynamic programming framework which allows defining the interactions between smartphones and smart things by means of on-the-fly scripts [13]. The scripts are executed in the smartphone, and they make use of the virtual profile stored in it for reconfiguring the behavior of the smart thing with the information available.

Our programming framework is inspired by the vision of a Programmable World [17], which foresees the evolution from today's IoT based on data recollection to truly programmable devices. This way, both smart things and smartphones are able to learn from each other, and to evolve through each interaction in a transparent and dynamic way.

In this paper, we present a formal framework for Digital Avatars. The framework provides a formal description of virtual profiles and the scripts to execute on them, and it establishes the basis for issues like privacy or security, with secure connections controlling the access to virtual profiles. The formalization is based on a multiple shared tuple spaces model inspired by Linda, which makes possible to ensure the soundness of the framework, and makes feasible the analysis of some interesting properties.

The rest of this paper is structured as follows. In Section 2 we present the motivations of this paper and discuss some related works. Next, Section 3 defines the concepts necessary to reason on Digital Avatars. In Section 4, we formalize the interactions that take place in the framework and demonstrate interesting formal properties of the system. Then, Section 5 presents a proof of concept and analyze its implementation using the framework. Finally, Section 6 draws the conclusions of the paper and briefly discusses future work.

2 Background

The development of smart things is transforming people's lives, as we increasingly interact with them everyday. Social Computing (SC) [18] is the area of computer science that deals with the interaction between social behavior and computer systems. SC encompasses all those systems that collect, process and disseminate information related to individuals and groups of people. The goal is learning about people and their preferences and providing an easy adaptation of their IoT environment, reducing manual configuration of devices to a minimum. Indeed, a number of recent research works agree on giving support to the IoT by means of a paradigm focused on people [16].

Currently, very few companies are able to access and process this enormous quantity of social information, and to exploit and make a profit from it. In practice, this reduces the SC marketplace to a small number of big stakeholders. As Tim Berners-Lee declared recently [1], SC systems should empower people, making them the fair owners of their information, and deciding who has access to it. Moreover, this information must be stored in a unique and accessible place which lets third parties use it in a controlled way, following the privacy preferences of the users.

In this same sense, we advocate for developing collaborative architectures based on smartphones. Their pervasive presence in people's everyday lives and their increasing sensoring and computing capabilities, together with their communication skills, make them key elements for obtaining, processing, and sharing information about their users [15]. Smartphones are also the most appropriate devices to be in charge of negotiating the interactions of their users with smart things in their environment.

Architectures based on P2P models are gradually acquiring a greater presence in fields such as social networks [19] or recommendation systems [20]. The basis of these architectures are the virtual profiles of the users, plenty of contextual data (e.g. activities, relations with other users, etc.) [8]. Our goal is sharing these virtual profiles with third parties and to adapt the IoT environment to the preferences and needs of each user.

With that purpose in mind, our approach is based on a Linda-like model. Linda [7] is a coordination language where synchronization is achieved by means of a shared tuple space, and through a set of simple but enough expressive primitives [2]. However, a single shared tuple space would violate the principles of the PeaaS model. Some other Linda-like proposals have been made by different authors, introducing some kind of mobility, mainly based on adding capabilities for remotely modifying a given tuple space. Thus, Lime [14] was proposed as a Linda extension to support mobile computing, by the definition of transiently shared distributed tuple spaces to establish P2P communications. Some of its goals are common with ours, but our framework also takes into account privacy issues, which are crucial for virtual profiles. Another well-known proposal is KLAIM [5], which extends Linda by considering the possibility of remote adding tuples to an *accessible* tuple space. With a similar philosophy, SCEL [6] was designed to provide a parametric language to capture various programming abstractions for autonomic components and their interaction. In both cases, Linda-like primitives were added to allow the remote interaction with shared tuple spaces. Although the PeaaS paradigm could be (artificially) coded by these languages, a number of assumptions and constraints should be made to ensure the main PeaaS features. In fact, we consider that accessing to a virtual profile has to be made only by its owner, and remote accessing to transient tuple spaces or shared repositories do not model these scenarios properly.

3 Modeling Digital Avatars

In order to define a formal framework for reasoning on Digital Avatars, we introduce the notion of virtual profile together with a number of related concepts, and we describe how virtual profiles can be offered as services under the PeaaS paradigm.

3.1 Definitions

The key issue for taking into account the user in an IoT environment is her virtual profile. It contains information about user preferences, habits, movements, or relations. All this information is only stored in the user's device (e.g. a smartphone), and it is offered as a service to third parties. The definition below formalizes this notion.

Definition 1. *A* virtual profile P *is a multiset of entities, where each entity is a 5-tuple* $t = (n, s, p, v, ts)$ *composed of (i)* $n \in \mathit{Name}$ *representing the name of the entity, (ii)* $s \in \mathit{Type}$ *defines the entity's type, (iii)* $p \in \mathit{Privacy}$*, which provides the level of privacy, (iv)* $v \in \mathit{Value}$ *is the value of the entity itself, with a structure which will depend on the entity's type, and (v) a timestamp* $ts \in \mathit{Time}$*, which allows recording the time when the tuple is added to the virtual profile. We will denote by* T *the set of tuples, and by* $\mathcal{P}$ *the set of virtual profiles.*

The complexity of virtual profiles depends on the sets *Name*, *Type*, *Privacy*, *Value*, and *Time*. These entities are structured in nested sections for the secure and correct functioning of the profile. Although the model does not depend on how these particular domains are defined, we consider a common minimum structure for predefined entities which are characterized as follows:

Personal It consists in personal and contact information of the user (`personal` $\in$ *Name*); the default privacy is `private` $\in$ *Privacy*, although it can be overwritten in each attribute to allow accessing it to family or friends, for instance. Basically, it contains a collection of entity identifiers like `name`, `phone`, `address`, or `email` with the corresponding information.

Relations It provides information on how users are related to each other (`relation` $\in$ *Name*), such that values includes entity names like `family`, `friends`, `colleagues`, or `acquaintances`. These nested entities are collections of user (personal) information with information about location, social profiles and their certificate hash fingerprint. The default privacy level of these entities is `private` $\in$ *Privacy*.

Places It defines information in a profile concerning locations (`place` $\in$ *Name*): home, place of work, known places or other places. Thus, constants like `home` or `work` belong to the *Name* set in the value of this entity. Their default privacy level is `trusted`.

A virtual profile can be accessed and/or modified by means of processes executing appropriate actions. In order to formalize this idea, we are inspired by

Linda [4], a coordination language [7] consisting of a set of inter-agent communication primitives, which can be virtually added to any programming language. Primitives in Linda allow processes to read, delete, and add tuples in a shared *tuple space*. Tuple spaces are a convenient approach to represent virtual profiles shared by concurrently running processes. A virtual profile is represented by a multiset of tuples encapsulated in a device. Thus, we adopt a multiple tuple space model.

Following other approaches [3,12], we shall consider a process algebra $\mathcal{L}$ including the Linda communication primitives and the usual concurrency connectives, parallel and non-deterministic choice. The primitives permit to add a tuple (*out*), to remove a tuple (*in*), and to check the presence (or absence) of a tuple (*rd*, *nrd*) in a given profile (tuple space).

Processes in $\mathcal{L}$ provide a convenient way to model scripts which can be downloaded from a server and run on a smart device. Thus, the syntax of $\mathcal{L}$ is formally defined as follows:

$$\begin{aligned} S \in \mathcal{L} &::= 0 \ \mid \ \alpha.S \ \mid \ S + S \ \mid \ S \parallel S \ \mid \ S(\tilde{t}) \\ \alpha \in Act &::= rd(t) \ \mid \ nrd(t) \ \mid \ in(t) \ \mid \ out(d,t) \end{aligned}$$

where 0 denotes the empty process, $d \in D$ a device identifier, and t denotes a tuple. The process $S(\tilde{t})$ denotes a procedure call where the procedure definition will be given by a script template $S(\tilde{x})$ (where $\tilde{x}$ is a sequence of variables instantiated by a sequence of tuples $\tilde{t}$). In order to simplify the definition of rules modelling the $\mathcal{L}$ primitive actions in Subsection 4.2, we will assume that reading a tuple do not imply the evaluation of usual operations (e.g. arithmetic operations) nor the variable instantiation as usual in Linda-based languages. This assumption does not imply any loss of generality of the proposal.

Notice that we consider primitives for locally accessing, adding, and removing tuples to a tuple space (i.e. a virtual profile). Although we could have also considered accessing and deleting tuples from remote tuple spaces, for our purposes we only need to add tuples remotely. For this reason, only the *out* primitive includes as a parameter the device on which adding the tuple. That is, *rd*, *in* and *nrd* actions will be made locally, on the same device where the script is being run. The same considerations were made in [12]. As it will be shown later, remote adding of tuples will only affect to the artifact where the script was downloaded from, thus we will not allow arbitrary remote adding of tuples. This asymmetric treatment of out and read primitives are precisely one of the features devoted by the PeaaS model: local accessing is only made by device owners, and remote changes can only be made on artifacts providing the scripts to be run.

In our framework, we distinguish two kinds of artifacts: *smart devices* and *smart things*. The difference between them is that smart devices exhibit computing capabilities, and therefore they can download and execute scripts, whereas smart things only provide a (link to a) script.

Formally, we define an artifact as a pair consisting of a virtual profile and a process corresponding to the execution of one or several scripts. We assume that D is a set of artifact identifiers. Every artifact d also has associated a script definition $S_d(\tilde{x})$ which can be downloaded by other artifacts with computing capabilities (i.e. smart devices).

Definition 2. *An* artifact $d \in D$ *is characterized by a pair* $\langle P : S \rangle_d$*, including a virtual profile* P *and a process* $S \in \mathcal{L}$*, corresponding to the running scripts on the artifact. In addition, an artifact can contain a script definition* $S_d(\tilde{x})$*. We will denote by* $S_d(P)$ *the script instantiated by the specific tuples in the profile* P*. And we will represent by* $\mathcal{D} = \mathcal{P} \times \mathcal{L} \times D$ *the set of artifacts.*

A smart thing will be characterized by having only a profile; that is, its process is always the empty process 0. A typical example of smart thing would be a beacon broadcasting a Bluetooth Low Energy (BLE) signal which encodes the URL of a script file to be downloaded from a server. On the other hand, typical smart devices are smartphones, tablets, or any other device with computing capabilities. Both kinds of artifacts —smart things and smart devices— store information in a virtual profile.

3.2 Security in Digital Avatars

The actions executed over the virtual profile of a smart device may emerge from internal processes of the device, or they may be part of a script downloaded from another artifact (e.g. a beacon broadcasting a link to a script file) when several conditions are fulfilled: the smart device is close enough to the beacon, the beacon artifact is registered, its script code is trusted, etc.

In order to avoid running untrusted scripts, we assume a Certification Authority capable to ensure the trustfulness of an artifact d, and a Boolean mapping *certify* which provides this information in such a way that *certify*(d) is true when the emitter of d has been authenticated.

In addition, the *out* primitive considered in the previous section allows adding tuples to both local and remote virtual profiles. Although the model imposes no limitations on which profiles can be remotely modified, the artifact identifier d used in a remote *out(d,t)* in a script $S_d(\tilde{x})$ can only be that of the artifact d itself. Thus, an artifact's profile may only be remotely modified by running a script downloaded from this same artifact.

Hence, in order to guarantee that the actions executed while running a script on a smart device are secure, we assume a Boolean mapping $\mathit{accept} : \mathit{Act} \times \mathcal{P} \rightarrow \{\mathit{true}, \mathit{false}\}$ that restricts which primitives are enabled, in such a way that $\mathit{accept}(\alpha, P)$ is true when the action α is acceptable on the profile P.

However, using certificates and restricting remote addition of tuples is not enough to ensure a correct interaction between source and target artifacts, and we also need to consider some technical issues. Indeed, whereas *certify* provides a third-party declaration about the trust of an artifact, and *accept* controls what actions are permitted inside an artifact once the script has been downloaded, we need a way to detect when two artifacts are actually able to communicate with each other. For instance, consider a scenario where a smartphone (represented by a virtual profile P) approaches a smart thing d which provides a script $S_d(\tilde{x})$. For downloading the script from d and running it in the smartphone, we assume a mapping $\mathit{links} : \mathcal{P} \times D \to 2^T$, which provides a link to connect to the smart thing, depending on the availability to download, the closeness between both artifacts, good signal strength, etc. This mapping returns a set of tuples representing links (e.g., a URI or a bluetooth connection) providing a way to access the artifact d. If there are no links, or the profile P does not accept downloading the script offered by d, $\mathit{links}(P, d)$ will be the empty set.

Notice that all the notions introduced in this subsection (*accept*, *certify*, and *links*) are application specific, in such a way that their particular definitions will depend on the application domain and context where our framework is applied to.

4 Formal framework

Now that we have defined the main elements and concepts of our framework, we can formalize the interactions between artifacts by means of a transition system with in-device and remote operations. Then, we show how some interesting properties like bisimilarity and congruence are accomplished by the model.

4.1 In-device transition system

The operational semantics of $\mathcal{L}$ is modeled by the following labelled transition system:

$$\stackrel{\cdot}{\longrightarrow} \subseteq \mathcal{D} \times \Lambda \times \mathcal{D}$$

defined by the rules [1] of Table 1, where $\mathcal{D} = \mathcal{P} \times \mathcal{L} \times D$ and $\Lambda = \{t, \bar{t}, \underline{t} : t \in T\} \cup \{\tau\}$.

Rule $\textsc{Out}_1$ describes how the output operation proceeds as an internal move (represented by label τ) which adds the tuple t to the profile P (comma is used to represent the multiset union). Rule $\textsc{Out}_2$ shows that a tuple t is ready to offer itself to the artifact/device by performing an action labelled $\bar{t}$. Rules $\textsc{In}$ and $\textsc{Read}$ describe the behavior of the prefixes $in(t)$ and $rd(t)$ whose labels are $\underline{t}$ and t, respectively. Rule $\textsc{NRead}$ describes the prefix action $nrd(t)$, which proceeds when t is not in the profile P; the transition is labelled with $\neg t$. All these rules need that the current device's profile accepts the corresponding action. It is worth noting that we do not include any kind of evaluation nor variable instantiation when reading tuples, as it is usually made in Linda-related transition rules. This is only for simplicity reasons without loss of generality.

Rule $\textsc{Sum}$ is the standard rule for choice composition. Rule $\textsc{Sync}_1$ is the standard rule for the synchronization between the complementary actions t and $\bar{t}$. It models the effective execution of an $rd(t)$ operation. Notice that the resulting profile is left unchanged, since the read operation $rd(t)$ does not modify it. Rule $\textsc{Sync}_2$ defines the synchronization between two processes performing transitions labelled with $\underline{t}$ and $\bar{t}$, respectively. It models the effective execution of $in(t)$

[1] For the sake of simplicity we will consider only finite processes here.

action. The usual rule $\textsc{Par}_1$ for the parallel operator can be applied to any label. The transition system is considered closed w.r.t. commutative and associative properties for sum $(+)$ and parallel $(\|)$ operators.

$(\textsc{Out}_1)$	$\dfrac{accept(out(d,t),P)}{\langle P : out(d,t).S\rangle_d \xrightarrow{\tau} \langle P, t : S\rangle_d}$
$(\textsc{Out}_2)$	$\dfrac{}{\langle P, t : S\rangle_d \xrightarrow{\bar{t}} \langle P : S\rangle_d}$
$(\textsc{Read})$	$\dfrac{accept(rd(t),P)}{\langle P : rd(t).S\rangle_d \xrightarrow{t} \langle P : S\rangle_d}$
$(\textsc{In})$	$\dfrac{accept(in(t),P)}{\langle P : in(t).S\rangle_d \xrightarrow{\underline{t}} \langle P : S\rangle_d}$
$(\textsc{NRead})$	$\dfrac{t \not\in P \ \wedge \ accept(nrd(t),P)}{\langle P : nrd(t).S\rangle_d \xrightarrow{\neg t} \langle P : S\rangle_d}$
$(\textsc{Sum})$	$\dfrac{\langle P : S_1\rangle_d \xrightarrow{\alpha} \langle P' : S_1'\rangle_d}{\langle P : S_1 + S_2\rangle_d \xrightarrow{\alpha} \langle P' : S_1' + S_2\rangle_d}$
$(\textsc{Sync}_1)$	$\dfrac{\langle P : S_1\rangle_d \xrightarrow{t} \langle P : S_1'\rangle_d \ \langle P : S_2\rangle_d \xrightarrow{\bar{t}} \langle P' : S_2\rangle_d}{\langle P : S_1 \parallel S_2\rangle_d \xrightarrow{\tau} \langle P : S_1' \parallel S_2\rangle_d}$
$(\textsc{Sync}_2)$	$\dfrac{\langle P : S_1\rangle_d \xrightarrow{\underline{t}} \langle P : S_1'\rangle_d \ \langle P : S_2\rangle_d \xrightarrow{\bar{t}} \langle P' : S_2\rangle_d}{\langle P : S_1 \parallel S_2\rangle_d \xrightarrow{\tau} \langle P' : S_1' \parallel S_2\rangle_d}$
$(\textsc{Par}_1)$	$\dfrac{\langle P : S_1\rangle_d \xrightarrow{\alpha} \langle P' : S_1'\rangle_d}{\langle P : S_1 \parallel S_2\rangle_d \xrightarrow{\alpha} \langle P' : S_1' \parallel S_2\rangle_d}$

Table 1. Transition system for smart devices

Notice that action $out(d,t)$ is only considered in Table 1 when it is running in the device d. Its full behavior (remote adding of tuples) will be defined when the interaction among devices is expressed in Table 2.

4.2 Remote transition system

In order to define how artifacts interact, we consider configurations composed of a parallel composition of artifacts as follows:

$$\langle P_1 : S_1\rangle_{d_1} \mid \langle P_2 : S_2\rangle_{d_2} \mid \cdots \mid \langle P_n : S_n\rangle_{d_n}$$

where P_i $(i = 1..n)$ are virtual profiles of artifacts —either smart devices and smart things—, S_i are scripts running in smart devices, and d_i represent the device identifiers. Notice that we denote in a different way the parallel composition of artifacts ($|$) and the parallel composition of processes inside a smart device ($\|$).

The transition system $\xrightarrow{\cdot}$ defined in Table 1 is extended to configurations by the inference rules given in Table 2.

$$\text{(Remote)}\quad \frac{accept(out(e,t),Q)}{\langle P : out(e,t).S\rangle_d \mid \langle Q : T\rangle_e \xrightarrow{\tau} \langle P : S\rangle_d \mid \langle Q, t : T\rangle_e}$$

$$\text{(Sync}_3\text{)}\quad \frac{certify(e) \wedge b \in links(P,e)}{\langle P : S\rangle_d \mid \langle Q : T\rangle_e \xrightarrow{\tau} \langle P, b : S \parallel S_e(b,Q)\rangle_d \mid \langle Q : T\rangle_e}$$

$$\text{(Par}_2\text{)}\quad \frac{D_1 \xrightarrow{\alpha} D_1'}{D_1 \mid D_2 \xrightarrow{\alpha} D_1' \mid D_2}$$

Table 2. Transition system

Rule Remote models remote actions modifying the virtual profile which belongs to the smart thing from which the script being run was downloaded. We consider this transition as a silent step from an observational point of view. For this reason, we use the label τ.

Rule Sync$_3$ represents the interaction between two artifacts (typically, a smart device and a smart thing). In this case, the script associated with a smart thing e, previously certified, is downloaded through a link b establishing a connection between the virtual profile P and e. Thus, the script to be executed in the context of the smart device d (in parallel with other possible pending processes) will be $S_e(b,Q)$ (such as it was defined in Definition 2). Notice that, in this case, the script is instantiated not only by the profile Q but also by the link b. This allows customizing the script to the artifact which provides access to it. In addition, the link tuple b is added to the profile P, so recording that the smart thing has been already "visited".

Rule Par$_2$ describes the way in which the parallel composition of artifacts proceeds. Note that the parallel composition of processes inside a smart device is modelled by Rule Par$_1$ in Table 1. Actually, any interaction in the context of a smart device is governed by rules in that table.

We consider the transition system closed w.r.t. usual structural congruence (commutative and associative properties) of both parallel connectors.

The rules in Table 1 and Table 2 are used to define the set of derivations in an environment where smart devices and smart things are interacting with each other. Following [3], both reductions labelled τ and reductions labelled $\neg t$ are considered. Formally, this corresponds to introducing the following derivation relation:

$$D \longmapsto D' \quad \text{iff} \quad (D \xrightarrow{\tau} D' \text{ or } D \xrightarrow{\neg t} D').$$

4.3 Bisimilarity and congruence

The scripts downloaded from a given artifact may evolve under certain circumstances. For instance, a software upgrade, or the development of a new version of the script. In this kind of situations, a notion of script equivalence would be very relevant to reason about compatibility among different versions. To formalize this, we consider the usual notion of bisimilarity-based equivalence, taking into account the device in which the script has to be run.

Definition 3. *Given a virtual profile P, two scripts S and T in $\mathcal{L}$ are bisimilar with respect to P, written $S \sim_P T$ if and only if for each $\alpha \in \Lambda$ and $d \in D$:*

1. *if $\langle P : S\rangle_d \xrightarrow{\alpha} \langle P' : S'\rangle_d$ then $\langle P : T\rangle_d \xrightarrow{\alpha} \langle P' : T'\rangle_d$ for some T' such that $S' \sim_{P'} T'$*
2. *if $\langle P : T\rangle_d \xrightarrow{\alpha} \langle P' : T'\rangle_d$ then $\langle P : S\rangle_d \xrightarrow{\alpha} \langle P' : S'\rangle_d$ for some S' such that $S' \sim_{P'} T'$*

Lemma 1. *The bisimilarity relation $\sim_P$ is an equivalence relation.*

Proof. It is directly derived by reasoning on different rules in Table 2. □

In fact, the transition relation $\stackrel{\cdot}{\longrightarrow}$ (restricted to devices) defines a notion of bisimilarity which permits to decide about script equivalence. In addition, it would be very useful that this bisimilarity relationship is a congruence with respect to the connectors $+$ and $\|$.

Theorem 1. *The bisimilarity relation $\sim_P$ is a congruence with respect to non-deterministic choice and parallel operators.*

Proof. Let P be a virtual profile, and let us assume $S_1 \sim_P S_2$. We will prove $S_1 \parallel T \sim_P S_2 \parallel T$ by structural induction. To do it, we will only analyze the first condition in Definition 3, since the second one is symmetric. That is,

$$\langle P : S_1 \parallel T\rangle_d \xrightarrow{\alpha} \langle P' : S'\rangle_d \tag{1}$$

First, we proceed by proving the proposition on the inductive base, processes 0 and $\alpha.0$ ($\alpha \in Act$). For these processes, the result is easily proved, by considering each case in rules (Out$_1$), (Out$_2$), (Read), (In) and (NRead).

In a general case, we have the following alternatives (depending on the rule in Table 1 triggering the transition):

1. If (Par$_1$) was the rule applied to get (1), then we have two possibilities: either

$$\langle P : S_1\rangle_d \xrightarrow{\alpha} \langle P' : S_1'\rangle_d \quad (S' = S_1' \parallel T)$$

or

$$\langle P : T\rangle_d \xrightarrow{\alpha} \langle P' : T'\rangle_d \quad (S' = S_1 \parallel T')$$

In the first case, as $S_1 \sim_P S_2$, we have $\langle P : S_2\rangle_d \xrightarrow{\alpha} \langle P' : S_2'\rangle_d$, with $S_1' \sim_{P'} S_2'$. Therefore, in both cases, by applying rule (Par$_1$):

$$\langle P : S_2 \parallel T\rangle_d \xrightarrow{\alpha} \langle P' : S''\rangle_d$$

S'' being $S_2' \parallel T$ or $S_1 \parallel T'$, respectively. Then, by applying inductive hypothesis on the first case $S' = S_1' \parallel T \sim_{P'} S_2' \parallel T = S''$, and $S'' = S'$ (hence $S'' \sim_P S'$ by Lemma 1 in the second one.

2. If the applied rule to get (1) is (Sync$_1$), then $\alpha = \tau$, $P' = P$, and either S_1 or T is a parallel composition of processes T_1 and T_2 such that T_1 implies a transition labelled by t. If $T = T_1 \parallel T_2$, we would have in the previous alternative (1). So, let's suppose $S_1 = T_1 \parallel T_2$, and

$$\langle P : T_1\rangle_d \xrightarrow{t} \langle P : T_1'\rangle_d \quad \langle P : T_2 \parallel T\rangle_d \xrightarrow{\bar{t}} \langle P' : T_2 \parallel T\rangle_d$$

with $S' = T_1' \parallel T_2 \parallel T$. By applying rule (Par$_1$) to the left transition above, we have $\langle P : S_1\rangle_d \xrightarrow{t} \langle P : T_1' \parallel T_2\rangle_d$. As $S_1 \sim_P S_2$, we have

$$\langle P : S_2\rangle_d \xrightarrow{t} \langle P : S_2'\rangle_d$$

for some S_2' with $S_2' \sim_P T_1' \parallel T_2$. Taking into account that transition $\bar{t}$ only affects to the profile P, we also have $\langle P : T\rangle_d \xrightarrow{\bar{t}} \langle P' : T\rangle_d$. Therefore, rule (Sync$_1$) applied to $S_2 \parallel T$ gets

$$\langle P : S_2 \parallel T\rangle_d \xrightarrow{\tau} \langle P : S_2' \parallel T\rangle_d$$

At this point $S' = T_1' \parallel T_2 \parallel T$ and $S_2' \sim_P T_1' \parallel T_2$, which implies (again by inductive hypothesis) $S' \sim_P' S_2' \parallel T$.

3. The last alternative to get transition (1) is applying rule (Sync$_2$). The reasoning is similar to the previous one.

In a similar way, we could prove $S_1 + T \sim_P S_2 + T$ when $S_1 \sim_P S_2$. □

5 Case study: a Treasure Hunt

Now that we have formally defined our framework for reasoning on Digital Avatars, we present a motivating example for showing how it works, and we discuss how the formalization provides useful tools for checking properties and inferring results of the systems built according to our proposal.

For that, we have implemented a treasure hunt game, in which several players look for a set of five hidden treasures following clues. Each treasure found provides a clue for a new treasure, and the player that first finds all the treasures wins the game. Treasures are represented by beacons, scattered over the scenario of the treasure hunt. Each beacon emits a BLE signal with the URL of a script file. When a player gets close enough to a beacon, her phone detects it, and downloads and runs the script.

For implementing the game, we only need to define one script template TH for all the beacons. This script is stored in a server which also keeps a virtual profile containing tuples with the clues for the treasures, and two extra tuples for the global state of the game, as it will be explained below. Prior to each download, the script is instantiated (Code 1) with the clues for finding five treasures (binding `_Clue1`, ..., `_Clue5` by the read actions in lines 1–3), and the current status of the game, which depends on whether there exists a `gameover` tuple (lines 4–5) or not (lines 7–8).

```
1  rd(<clue1,_Clue1>). rd(<clue2,_Clue2>).
2  rd(<clue3,_Clue3>). rd(<clue4,_Clue4>).
3  rd(<clue5,_Clue5>).(
4      rd(<gameover>).
5      TH(<gameover>,_Clue1,_Clue2,_Clue3,_Clue4,_Clue5)
6      +
7      nrd(<gameover>).
8      TH(<playing>,_Clue1,_Clue2,_Clue3,_Clue4,_Clue5) )
```

Code 1. Script instantiation.

The execution of the script interacts with the virtual profile in the smartphone, checking and updating which treasures have been already found by the player, and showing the clue for a new treasure. It also informs the game when a player has found all the treasures. The rest of the players will be notified the next time they find a beacon. The full script is shown in Code 2.

```
1  TH(_Status,_Clue1,_Clue2,_Clue3,_Clue4,_Clue5) =
2      nrd(<clue,C>).
3          out(<clue,_Clue1>). out(<clue,_Clue2>).
4          out(<clue,_Clue3>). out(<clue,_Clue4>).
5          out(<clue,_Clue5>). out(<treasures,0>). 0
6      +
7      in(<clue,C>). out(<_Status>).(
8          in(<playing>).(
9              nrd(<treasures,4>).
10             in(<treasures,X>). out(<treasures,X+1>).
11             out(<notify,C>). 0
12             +
13             in(<treasures,4>). out(<treasures,5>).
14             rd(<personal.name,Me>).  rd(<system.now,Time>).
15             rout(<winner,Me,Time>). rout(<gameover>). 0 )
16         +
17         in(<gameover>).
18             out(<notify,"You lose!">). 0 )
```

Code 2. Script for the Treasure Hunt.

The branch starting from line 2 in the script is performed when a player begins the treasure hunt (we assume an additional beacon in the starting place), as no clues are present in his virtual profile yet (line 2). In that case, all the five clues are added to the virtual profile of the player (lines 3–5). The last tuple added in line 5 is `<treasures,0>`, indicating that no treasures have been found yet.

Alternatively (line 7), a random clue is read (and consumed) from the profile, and the current status of the game (`<gameover>` or `<playing>`) is written in the player's profile. Then, we have again two alternatives. Either the game is over (lines 17–18) and the player is notified of this fact, or it is still being played (lines 8–15). In the latter case, the tuple `<treasures,N>` stores the number of beacons that the player has already found. If they are less than four (i.e. `<treasures,4>` is not in the profile, line 9), then we increase the number of treasures found (line 10), and show a new clue to the player (line 11). On the contrary, if the player had already found four treasures (line 13), she wins the game (please recall that the script is executed whenever the player finds a beacon, which makes it the fifth one). In this case, both the name of the player and the current time are get from the player's profile (line 14), which are used to update the global state of the game. Indeed, two tuples are remotely added to the server from which the script has been downloaded: one with information on the winner, and the other one indicating that the game is over (line 15). It is worth noting that the remote out actions in line 15 only have one argument, instead of two arguments as defined in *Act* in Section 3, because the first argument implicitly corresponds to the server storing the script.

5.1 Reasoning on the example

The formalization of Digital Avatars presented in this paper allows us to reason on the behavior of our treasure hunt game. One of the properties that we may want to analyze is whether it is ensured that eventually someone wins the game. Indeed, this property can be proved with the Linda-based semantics presented in Section 4, just making a couple of basic assumptions. First, let us consider an initial configuration composed of a non-empty set of smart devices (players) and at least one smart thing (beacon) pointing to a server which contains the script. This configuration is represented by a parallel composition of all those artifacts:

$$C_0 = \Pi_{i=1}^{n} \langle P_i : 0 \rangle_{d_i} \mid \langle P : 0 \rangle_b \tag{2}$$

being d_i the smart devices, each with a profile P_i, and b the beacon associated with a server with a profile P and script template S_b as specified in Code 2.

Second, let us assume that the smart thing is certified (i.e. *certify*(b)) and that all the devices d_i can always get a link to the beacon b. In other words, let $link(P_i, b) \neq \emptyset$ for an unlimited number of times, and for every $i = 1..n$. These two assumptions are formalized as hypothesis of the next proposition, which ensures the eventual end of game.

Proposition 1. *Let us consider a smart thing b containing the script template S_b as defined in Code 2, and the initial configuration C_0 specified in equation (2),* *such that accept(a, P_i) for every action a in any instance of S_b, and certify(b). If for every sequence of transitions $C_0 \longmapsto C$, some of the smart devices d_k in C has a virtual profile Q_k which satisfies link$(Q_k, b) \neq \emptyset$, then there exists a trace*

$$C_0 \longmapsto C' \mid \langle t, P' : 0\rangle_b$$

with t=`<gameover>`*.*

Proof. Applying the assumption to the empty sequence of transitions $C_0 \longmapsto C_0$, we can find a device d_{k_1} such that rule $\textsc{Sync}_3$ can be applied, because both conditions of that rule are fulfilled: *link*$(P_{k_1}, b) \neq \emptyset$ and *certify*(b). Therefore, running the script on that device, after applying several times the rules of Table 2, we obtain a trace:

$$C \longmapsto C_1$$

where a new configuration C_1 is achieved containing a device d_{k_1} whose profile includes the tuple `<treasures,1>`. If we apply again the hypothesis to C_1, we find a device d_{k_2} such that rule $\textsc{Sync}_3$ may be applied once more, and a new instance of the script S_b will make D_1 to progress to C_2 ($D_1 \longmapsto D_2$). If $d_{k_2} = d_{k_1}$, its profile will include the tuple `<treasures,2>`. If this is not the case, then we will have a new device with a profile also including `<treasures,1>`. Taking into account that we can always repeat this procedure, and that we have a finite number (n) of smart devices, after at most $4n$ iterations, some of the devices will exhibit a profile with the tuple `<treasures,4>`. Hence, the branch represented by lines 13–15 of S_b will be eventually triggered, adding the tuple `<gameover>` to b's profile. □

The proposition above shows that, under some basic assumptions, a proper initial configuration will eventually progress to a gameover status.

Additionally, some unexpected scenarios can be detected if we analyze the script in Code 2 more deeply. An exhaustive exploration of all possible traces generated from a configuration C, like in equation 2, would provide interesting information about the soundness of the script. In fact, a model checker capable of exhaustively exploring all possible traces achievable from C would detect some target configurations such that $C \longmapsto C' \mid \langle P' : 0\rangle_b$, where the profile P' includes two or more copies of tuple `<winner,Me,Time>`. This means that two or more players could postulate themselves as winners of the treasure hunt. Indeed, it is easy to imagine how C can progress to a configuration D, such that:

$$D = \ldots \mid \langle P_d : S_d\rangle_d \mid \langle P_e : S_e\rangle_e \mid \ldots \mid \langle P : 0\rangle_b \qquad (3)$$

where `<treasures,4>` is both in the profiles P_d and P_e, and `<playing>` is still in P. In other words, two devices would have found four clues each, and the game is still being played. In this situation, if we consider a scenario where $links(P_d, b)$ and $links(P_e, b)$ are not empty, then two consecutive transitions can occur:

$$\begin{aligned} D &\xrightarrow{\tau} \ldots \mid \langle P_d : S_d \parallel S_b(P, l_d)\rangle_d \mid \langle P_e : S_e\rangle_e \mid \ldots \mid \langle P : 0\rangle_b \\ &\xrightarrow{\tau} \ldots \mid \langle P_d : S_d \parallel S_b(P, l_d)\rangle_d \mid \langle P_e : S_e \parallel S_b(P, l_e)\rangle_e \mid \ldots \mid \langle P : 0\rangle_b \end{aligned}$$

Both transitions are a consequence of applying rule $\textsc{Sync}_3$, where $l_d \in links(P_d, b)$ and $l_e \in links(P_e, b)$. The scripts downloaded from b, $S_b(P, l_d)$ and $S_b(P, l_e)$ are conveniently instantiated with information on profile P. As P still includes the tuple `<playing>`, both instances of S_b will add that tuple to the local profiles of d and e (line 7 in Code 2). And then, because `<treasures,4>` is present in both profiles P_d and P_e, the actions in lines 13–15 of Code 2 are executed:

$$\begin{aligned} C \longmapsto D &\longmapsto \langle P_d : S_d \parallel S_b(P, l_d)\rangle_d \mid \langle P_e : S_e \parallel S_b(P, l_e)\rangle_e \mid \langle P : 0\rangle_b \\ &\longmapsto \langle P'_d : S'_d\rangle_d \mid \langle P'_e : S'_e\rangle_e \mid \langle P' : 0\rangle_b \end{aligned}$$

where both P'_d and P'_e include the tuple `<treasures,5>`, and the hunt will be stopped by remotely adding (twice) the tuple `<gameover>` to P'. However, two remote out actions would have been previously made adding to P' two `<winner,M,T>` tuples, too, which of course is not what we would desire as the result of the game.

This phenomenon could be addressed in several ways. The first idea that may come up to us is implementing locks for controlling the atomic execution of the script, blocking it for downloading in a smart device until its execution in some other device ends. Another solution would be adding to the game a final podium stage, where the server announces the actual (photo finish) winner, depending on the time of each `<winner,M,T>` tuple. In any case, the formalization of the framework allows us to analyze the behavior of the system and to study these and other situations in order to detect problems and solve them.

6 Conclusions

In this paper, we have shown how the formalization of Digital Avatars by means of a multiple tuple space approach provides interesting tools to verify different properties of the system. We may validate the compatibility of a Digital Avatars system with different versions of a script proving bisimilarity and congruence properties. Moreover, we have shown that we may formally check the interactions in a Digital Avatars system, which gives us the opportunity of studying its correctness, proving desired (or undesired) properties, and studying what happens in different situations, like in our treasure hunt example ensuring that the game always ends, or whether they may exist multiple winners.

As for future work, we plan to extend the transition system in order to model the atomic execution of the scripts, avoiding this way the mutual exclusion issues previously mentioned. Furthermore, it would be necessary to build model checking tools for analyzing desired or undesired properties of the scripts.

References

1. Tim Berners-Lee. Solid. `https://solid.inrupt.com/`. Accessed: 2019-01-21.
2. Antonio Brogi and Jean-Marie Jacquet. On the expressiveness of coordination via shared dataspaces. *Sci. Comput. Program.*, 46(1-2):71–98, January 2003.

3. Nadia Busi, Roberto Gorrieri, and Gianluigi Zavattaro. On the Turing equivalence of Linda coordination primitives. *Electr. Notes Theor. Comput. Sci.*, 7:75, 1997.
4. Nicholas Carriero and David Gelernter. Linda in context. *Commun. ACM*, 32(4):444–458, April 1989.
5. R. De Nicola, G. L. Ferrari, and R. Pugliese. Klaim: a kernel language for agents interaction and mobility. *IEEE Transactions on Software Engineering*, 24(5):315–330, May 1998.
6. Rocco De Nicola, Diego Latella, Alberto Lluch Lafuente, Michele Loreti, Andrea Margheri, Mieke Massink, Andrea Morichetta, Rosario Pugliese, Francesco Tiezzi, and Andrea Vandin. *The SCEL Language: Design, Implementation, Verification*, pages 3–71. Springer International Publishing, Cham, 2015.
7. David Gelernter and Nicholas Carriero. Coordination languages and their significance. *Commun. ACM*, 35(2):96–, February 1992.
8. Tor-Morten Grønli, Gheorghita Ghinea, and Muhammad Younas. Context-aware and automatic configuration of mobile devices in cloud-enabled ubiquitous computing. *Personal and ubiquitous computing*, 18(4):883–894, 2014.
9. Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013.
10. Joaquin Guillen, Javier Miranda, Javier Berrocal, Jose Garcia-Alonso, Juan Manuel Murillo, and Carlos Canal. People as a Service: a mobile-centric model for providing collective sociological profiles. *IEEE software*, 31(2):48–53, 2014.
11. Dominique Guinard, Vlad Trifa, Friedemann Mattern, and Erik Wilde. From the Internet of Things to the Web of Things: Resource-oriented architecture and best practices. In *Architecting the Internet of Things*, pages 97–129. Springer, 2011.
12. Ronaldo Menezes, Andrea Omicini, and Mirko Viroli. On the semantics of coordination models for distributed systems: The LogOp case study. In *Foundations of Coordination Languages and Software Architecture (FOCLASA 2003)*, volume 97 of *Electronic Notes in Theoretical Computer Science*, pages 97–124. Elsevier, 2004.
13. Alejandro Pérez-Vereda, Daniel Flores-Martín, Carlos Canal, and Juan M Murillo. Towards dynamically programmable devices using beacons. In *International Conference on Web Engineering*, volume 11153 of *LNCS*, pages 49–58. Springer, 2018.
14. Gian Pietro Picco, Amy L. Murphy, and Gruia-Catalin Roman. Lime: Linda meets mobility. In *Proceedings of the 21st International Conference on Software Engineering*, ICSE '99, pages 368–377. ACM, 1999.
15. Mika Raento, Antti Oulasvirta, and Nathan Eagle. Smartphones: An emerging tool for social scientists. *Sociological methods & research*, 37(3):426–454, 2009.
16. Jorge Sa Silva, Pei Zhang, Trevor Pering, Fernando Boavida, Takahiro Hara, and Nicolas C Liebau. People-centric Internet of Things. *IEEE Communications Magazine*, 55(2):18–19, 2017.
17. Antero Taivalsaari and Tommi Mikkonen. A roadmap to the programmable world: software challenges in the IoT era. *IEEE Software*, 34(1):72–80, 2017.
18. Fei-Yue Wang, Kathleen M Carley, Daniel Zeng, and Wenji Mao. Social computing: From social informatics to social intelligence. *IEEE Intelligent systems*, 22(2), 2007.
19. Yufeng Wang, Athanasios V. Vasilakos, Qun Jin, and Jianhua Ma. Survey on mobile social networking in proximity (MSNP): approaches, challenges and architecture. *Wireless networks*, 20(6):1295–1311, 2014.
20. Wan-Shiou Yang and San-Yih Hwang. iTravel: A recommender system in mobile peer-to-peer environment. *Journal of Systems and Software*, 86(1):12–20, 2013.