On the Internet, Privacy and the Need for a New Architecture of Networked Information Services

Silvio Romero de Lemos Meira[1,3,4], Vanilson André de Arruda Burégio[1], Leandro Marques do Nascimento[1,2], Saulo Araujo[1]

[1] Centro de Informática, Universidade Federal de Pernambuco, www.cin.ufpe.br, Brazil.
[2] DEINFO, Universidade Federal Rural de Pernambuco. www.ufrpe.br, Brazil.
[3] CESAR – Recife Center for Advanced Studies and Systems, www.cesar.org.br, Brazil.
[4] Berkman Center for Internet and Society, Harvard University, cyber.law.harvard.edu, USA.

silvio@meira.com*, {vaab, lmn2, sma2}@cin.ufpe.br

{CAUTION: WORK STILL IN BETA MODE; FEEL FREE TO SEND US YOUR COMMENTS}

Abstract

It has recently hit the news that the US government might be collecting information about the networked behavior of millions of individuals, as part of the effort to preempt terror actions. If real, such massive collection of data is only possible because users have been herded into centralized information systems that are more easily tapped by all sorts of prying eyes. In the past, before the internet, it would not be possible to mine all physical items of personal communication of large populations, due to the sheer complexity and cost of it. But the past is gone and, for now, we propose a novel architecture for networked information systems, around the notion of SOCIAL MACHINES, making it far easier for users to control access to their data and, for most purposes, including government spying on individuals, significantly increase the cost and complexity of information gathering from personal sources if not authorized by their true owners.

Keywords: internet, privacy, Social Machines, information systems, programmable web.

1 Private data is not so private anymore

The problem: Personal use of internet services happens, for the most part and for the vast majority of users, on top of information systems that are provided for free, given that the user authorizes whoever offers the service to gather information about his behavior in order to sell it to third parties who, in turn, pay the providers for the service it gives its users for free. Such information gathering and selling cycle leads to massive data troves being stored and processed by most internet companies and, in turn, to the possibility that government agencies interested in the behavior of certain classes of individuals could, for a reasonable cost, gather data about all users of a given system.

According to sources [1][2], this is what has been happening under the umbrella of the PRISM program in the USA and there are reasons to assume that it could possibly the case in other countries that would have the same class of interest and means that the USA has for that end. So far, or before PRISM was made public, the issue of data ownership on networked information services was treated as a non-problem, with most users and providers thinking of it as peripheral or worse, a preoccupation of a few that had “something to hide”. Not anymore, even more so because the “nothing to hide” argument has long ago been dismissed as fragile [3], even naïve.

A further problem: The wealth of the data on individuals on the internet has increased by orders of magnitude recently, when social networks entered the picture. Before, data used to be alone, part of a single user's files. Now, with social networks like Facebook providing a myriad ways to connect and relate agents of all sorts, which use their services to interact and create meanings and knowledge, it is as if we had been preparing, over the last few years of social networks as means of mirroring the workings of society on the internet, the basis, also, for programs like PRISM do work as easily and transparently as it could be.

The consequence? "No Pay, No SLA": if the user is not paying for data storage and processing, plus creating and maintaining connections, relationships and interactions, she is the product, represented by her behavior, that will end up being sold on the network. She will be targeted by all sorts of agents that want and need to understand her behavior in their contexts. And that behavior is offered as a service, by the big (and small) internet companies to their clients. On the individual user's side, the providers try to assure us that we ourselves will never be identified, only our behavior, although it is clear that mashing up information from several internet sources that don't identify "you" is very likely to *identify you*, as voter targeting efforts of the last USA presidential campaign have shown [4][5].

The GRAND consequence: In a number of ways, you give the privacy of your behavior away in order to benefit from the *free* network effects offered by a wide range of service providers. One huge side effect of this consequence is that, under certain circumstances, governments can ask such service providers for all data relative to your online behavior. What is more, if the service complies, the cost of gathering information about your behavior drops to very close to zero. Of course, processing such data will cost much more, but much less than it would be the budget, in the past, to just gather an equivalent amount of data about the behavior of citizens, national and foreign. In fact, it can now be assumed that at least the US government has been trying to do so in a very large scale over the last few years and many other governments can also be thought to have that both desire and capability to do so.

Is that avoidable? Yes, but only if the way we design, implement and provide information services on the internet needs to be fundamentally redesigned, using foundations that center the network around individual agents, or users, and not in terms of massive information services. That's exactly what we will try to show in the remainder of this article.

2 Social Machines, an introduction

SOCIAL MACHINE (SM) is an informational paradigm that blends computational and social aspects into software [6]. It proposes a unified model to deal with the complexity of the emerging web around us, and a practical way to explain each and every entity connected to it [7]. We have been using the concept of SMs over the last two years as a generic, simple manner to describe, design, develop and deploy services like Twitter, Facebook and Dropbox as a combination of "machines" that 1. have a **behavior**, 2. **communicate** and 3. obey certain **rules** or **constraints**, while acting over internal and external information. Such machines –or their behavior, communication abilities and constraints– can be embedded in a conceptual abstraction model that, in its turn, can be viewed as a basic, "programmable", network building block.

2.1 The Social Machine Model

The Social Machine model has the traditional algorithmic Turing Machine model of computation as a basis and deals with possibly related and interacting building blocks as *Social Service Components* [8] that make use of notions from computing, communication (in the form of relationships and interactions) and control. Social Machines make use of *relationship* in order to weave "sociability" aspects into software, as a mean of dealing with the proliferation of the emerging "relationship-aware" applications and services on the Web of today. As stated in [8], a Social Machine can be defined as:

*"A connectable and programmable building block that wraps **(WI)** an information processing system **(IPS)** and defines a set of required **(RS)** and provided services **(PS)**, dynamically available under constraints **(C)** which are determined by, among other things, its relationships **(Rel)** with others."*

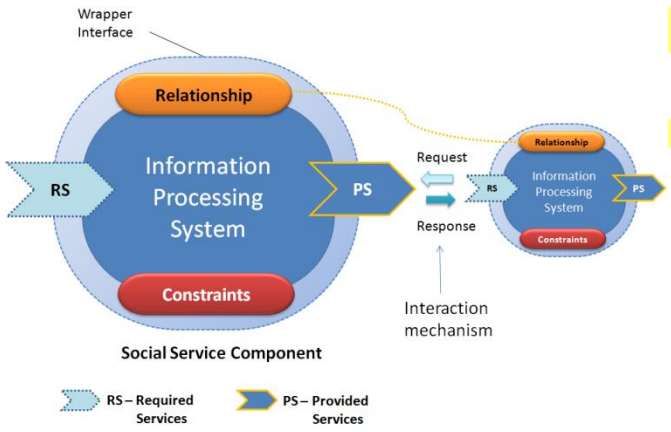

Fig. 1. A Social Machine as a Social Service Component.

Fig. 1 illustrates two interacting Social Machines and their main elements: information processing system, relationship, wrapper interface, provided services, required services and constraints. The notion of *information processing system* (IPS) abstracts any computational unit whose behavior is defined by the functional relationship between inputs and outputs. It can be a piece of either hardware or software. An IPS can be represented, for example, by an algorithm, a web service, a network of computer processes, or even person as a "unit of computation".

The notion of *Relationship* is the centerpiece of this model. It represents a particular type of connection that constrains the way of how two or more Social Machines are associated to or have interactions with each other.

The *wrapper interface*, in its turn, abstracts the communication layer through which a SM externalizes its services to allow interactions with others. In such layer, *provided services* (PS) represent the SM's business logic, dynamically offered as a set of services to other SMs, according to the *relationships* established between our SM and others that relate to it. Optionally, a Social Machine can specify *required services* (RS), which are whatever an SM needs to invoke in order to function properly.

And last, but not least, Social Machines have *constraints,* which are the rules for the establishment of *relationships* and definition of *interaction views* among different SMs.

2.2 Social Machines, some examples

The Social Machine model can be used to specify, for example, web services, but not only. As the paradigm has its origins on social computing [9], the early generation of Web-based social software (collectively called "Web 2.0) can also be considered primitive Social Machines; these are blogs, wiki-based systems, video sharing sites and so on. They have allowed users to interact and collaborate with each other by storing and sharing various types of content, including messages, photos and videos.

Social networking sites are also great examples of Social Machines, even more so because many of them (e.g. Twitter, Facebook) exposed, from the start, their internal capabilities as Web Services, in the form of open online application programming interfaces (*Open APIs*). These *Open APIs* can be seen as platforms which allow third-party developers to interact with social-networking sites, access information and media posted by their users and create other applications and services, on top of the platform, that aggregate, process, and generate content based on users' interests. That just may be the case in which *computing literally means connecting services* [10]. Several other examples of Social Machines, including *Systems based on Social Data*, *Crowdsourcing and Collaborative Platforms* and *Knowledge Acquisition Systems,* are discussed in [6], reinforcing the idea that Social Machine paradigm relies on social computing and can be viewed as the convergence of the three main approaches: *i) Social Software* (as foundation), *ii) People as Computational Units* and *iii) Software as Sociable Entities*. Additionally, an analysis describing Facebook as a relationship-aware Social Machine with 2^{82} interaction views is also reported in [8].

2.3 People as Social Machines

The previous sections show what Social Machines are all about, basically. Consequences? One of those "network building blocks" can represent **YOU** in a "system", for example. Thus, how to simulate a person as a Social Machine on the Web? One possibility is to consider persons as "units of computation", in the form of human-based computing [11], and *wrap* their information and/or human capabilities as services (e.g. APIs).

VoiceBunny[1] is a practical example of that. It is a crowdsourced platform that makes use of thousands of voice actors working from home studios to provide professional "*voice as a service*" on the Web. It offers RESTful APIs [12][13] through which is possible to create third-party applications that interact with and consume VoiceBunny's *provided services*.

The *Human API*[2] is another case of people as Social Machines. Its platform lets application developers create meaning from *personal data* (e.g., heart rate, active minutes, sleep, genetic makeup or blood glucose) through one simple API. Each data stream is exposed as an endpoint that can be called as a

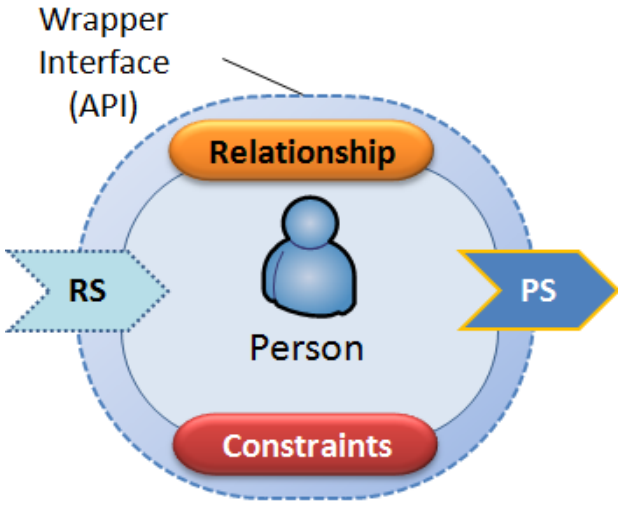

Fig. 2. Representation of a person as a Social Machine.

[1] Voice actors and professional voice over recordings (http://voicebunny.com).

[2] HumanAPI: A platform for human health data (http://humanapi.co).

service to build *mashups* that deal with human health data.

In each of the examples above, a person (i.e., his/her information and human capabilities) is *wrapped* to provide a set of services through simple, clean, stable APIs. Hence, we can generalize this idea to logically represent a person wrapped as a Social Machine, following the model described in this paper. **Error! Reference source not found.** illustrates this **logical view** of a person as Social Machine.

2.4 Social Networks: Federations of People as Social Machines? YES. HOW?

Social networks such as Twitter and Facebook have not only substantially changed the way we communicate, but also the way we develop software and how they operate and interact with each other. As aforementioned, modern social networking sites provide ways to access information of their users through open APIs that are made available as services on their platforms.

In practice, this enables the creation of a plethora of mashups (or API consumers) integrating data from one or more sources in order to build new applications combining (among other things) the users' information behavior. Such mashup ecosystem [14] is basically formed by networks of API providers and consumers.

Thus, adopting a logical view, every modern social network can be seen as a federation of [related and interacting] people as Social Machines. In such networks, each person (i.e., his/her related information) is wrapped by a communication layer (API) that allows access to their information and "social actions" by third-party mashups, as showed in **Figure 2**, below.

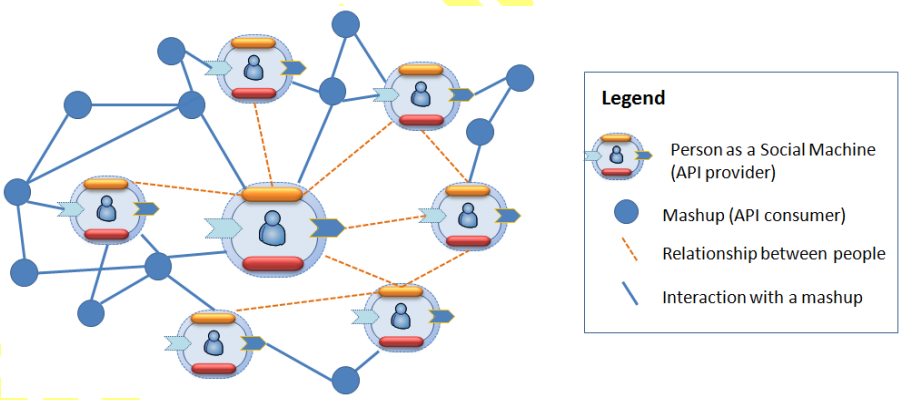

Fig. 2. Logical view of a Social Network as a Federation of People as Social Machines.

The view presented in **Figure 2**, however, is not clear for nearly every user. Indeed, from the user's perspective, very often we hear people asking questions like *"How come the newspaper's website is showing to my friends what I have read recently?"* In general, we do not know the way our information is manipulated and made available to the external world. So, as users, how could we control **WHERE**, by **WHOM** and **HOW** our information is stored, accessed, shared and combined?

We do not have answers to all these questions, but we have been trying to provide ways to make it easier for users to control their **OWN** data and, consequently, hide that from undesired onlookers. Next, we are going to describe that in a much more formal way that can lead to a software implementation and deployment of that idea.

3 How do we describe Social Machines in a more formal way?

As mentioned in different examples, the emerging web of Social Machines involves many ways of communication (such as different protocols and types of connections, in our context called *relationships*), many platforms and technologies, and also different programming languages. We notice that appropriate languages and tools may drastically reduce the cost of building new applications as well as maintaining existing ones [15] easing the task of formally describing Social Machines.

In the context of programming languages, one way to gain on expressiveness is by using a Domain-Specific Language (DSL), providing constructions and notation tailored towards a particular application domain [16]. Usually, DSLs are small, more declarative than imperative, and more attractive than General Purpose Languages (GPL) for a particular application domain. There are countless examples of DSLs for a lot of domains, as seen in [17], which shows a taxonomy of DSLs split into domains such as web [18], mobile apps [19], hardware description [20] and a category of DSLs that are of interest to us, here, Architecture Description Languages (or ADLs). According to [21], an ADL focuses on the high-level structure of the overall software application rather than the implementation details of any specific source module. In other words, ADLs focus on the high-level structure of software, while DSLs capture details of domains using specific notations that can be mapped to general purpose programming language constructs. However, there is no problem in using an ADL for a specific domain, with its respective level of details, as seen in [22] for the domain of real-time/embedded systems (automotive and avionics), and in [23] for the industrial applications domain.

We describe Social Machines using **SMADL – Social Machine Architecture Description Language**, mixing concepts of ADLs and DSLs to increase expressiveness. As an ADL, it allows the description of Social Machines in terms of relationships as high-level abstractions, without the need to specify details of communication (protocols), authentication and/or constraints for each relationship. As a DSL, it allows the implementation and integration of web services using a dynamically typed syntax, fully integrated to the Java Virtual Machine and Eclipse IDE. Also, if needed, Java expressions and Java pre-defined classes can be used as an option for implementing details of a given Social Machine.

3.1 Social Machines concepts described in SMADL

A Social Machine such as that of **Fig. 1** is defined in terms of *Relationships*, a *Wrapper Interface*, *Required Services* (RS), *Provided Services* (PS) and *Constraints*. **Table 1** shows the mapping of SM concepts into SMADL constructs.

Table 1. Mapping between Social Machine concepts and SMADL constructions.

Social Machine Concepts	SMADL Constructs
Social Machines	Social Machine entity
Relationships	Social Machine entity dependencies
Wrapper Interface	SM operations available under certain constraints over HTTP (later transformed to RESTful services)
Required Services (RS)	Relationship operations called from the current SM
Provided Services (PS)	Description of operations provided to general purpose
Constraints	General, Operational and Relationship Constraints

A *SocialMachine* entity can be defined in SMADL using the following syntax (details are presented next):

```
//facebook and dropbox must had been previously defined as
//Social Machine entities just like 'MyNewSocialMachine'
SocialMachine MyNewSocialMachine relates to facebook, dropbox {

  general constraint UNLIMITED

  Relationships {
     //SM 'dropbox' must be listed in the 'relates to' section
     dropBoxFiles with dropbox [
       uri = "http://localhost:8080/"
       api-key = "2190809345BJASDH"
       secret = "IQW098IUABGBAS"
       user-token = "18UYGS9876GU238"
     ] type: FULL_ACCESS //every single operation of dropbox

     //SM 'facebook' must be listed in the 'relates to' section
     facebookPosts with facebook [
       uri = "http://localhost:9091/"
       api-key = "IUASHKJHFOI99"
       secret = "0987ASIUH09YOIUHFA"
       user-token = "HJASDFHG82398BV87B"
     ] type: LIST_OF_OPS (post, listFriends)
  }

  constructor(String name, Integer initialPort) {
    var config = name //Constructor body (dynamically typed expression)
  }

  op listFilesInDropboxFolder
      returns java.util.List<java.io.File> (String folder) {
    return newArrayList //Example code (java based)
  } constraint PRE_AUTH_SM

  op createFacebookPost(String text) { /* Operation body */}
}
```

SMADL code snippet.

A SM entity is defined using scopes between curly braces, just like Java does. A typical SM entity has 4 sections: a (optional) general constraint, a group of Relationships, constructors and operations.

At the first line of an SM definition, the '`relates to`' section is optional. When present, this section lists other SMs to be used in `MyNewSocialMachine` scope. It is mandatory that the SMs listed in this section had already been defined in the same .smadl file or in the same project. This list will be used for defining every relationship of the current SM entity.

Constraints are used throughout all SM body and can be of three types: *general constraint* (example: '**`general constraint`**'), *operation constraints* (example: '**`constraint PRE_AUTH_SM`**') and *relationship constraints* (example: '**`type: FULL_ACCESS`**'). One SM can be used to provide and/or request services. When providing services, general and operation constraints are applicable. When requesting services, relationship constraints are applicable. There is no problem for a SM to request and provide services at the same time.

Every SM admits only one general constraint and it is applicable to all its operations ('**`op`**'). Operation constraints are valid for single operations, and, when present, they suppress a possible general constraint.

General and operations constraints can of four different types (relationship constraints are further explained):

- `UNLIMITED` - no limitations in SM service if any operation is called;
- `REQUESTS_PER_PERIOD` - counts the number of requests per period of time and blocks ;
- `PRE_AUTH_SM` - this constraint only allows the operation to be executed by a previously authorized SM;
- `REDUCED_RESOURCE` - this constraint does not count accesses made by other SMs, however it reduces the performance of resource delivery to whoever requests it. Defining a constructor for the entity is optional.

In the code snippet early showed, `MyNewSocialMachine` provides an operation named `listFilesInDropboxFolder`, which returns a list of files from Dropbox. After describing any operation, a specific constraint may be applicable, suppressing the early defined '`general constraint`'. In this case, an operation constraint is valid for all requests from any other SM, even if the requester is not part of the current SM relationships listed in '`relates to`' section. The descriptions of the operations (*Provided Services*) in addition to their respective constraints represent the *Wrapper Interface.*

3.2 Defining relationships and their correspondent operations

Relationships are the centerpiece of the Social Machine model. In SMADL, a relationship has one identifier (in the code snippet, the identifiers are `dropBoxFiles` and `facebookPosts`), its main parameters for establishing an OAuth connection, and the relationship type (or relationship constraint). These constraints for relationships can be of two types:

- **`FULL_ACCESS`** – when every operation of the target SM can be called by the current entity;
- **`LIST_OF_OPS`** – lists the operations permitted to be called by the current entity, for instance, if you are choosing to connect to Facebook and given that Facebook is already defined as a Social Machine using SMADL, then you will list the operations that can be used in the relationship between your current SM and the target. It means that while registering your application in Facebook developers' site, you will choose only the interactions needed, such as, posting new comments and listing friends

It is important to notice that, if any entities are listed in the '**`relates to`**' section, then it is mandatory to configure each appropriate relationship according to its particular constraint. It is not possible to call the operations of a "related" SM just by using its name without a relationship identifier. The identifier will provide the correct scope for calling the appropriate operations in that relationship.

4 Is a network of Social Machines less prone to prying eyes than Facebook or Gmail? Why?

We believe that a network of SMs is less prone to prying eyes than current approaches used by Facebook and others, based on the analysis of the following criteria: (1) *Vulnerability,* (2) *Heterogeneity*, (3) *Use of an Architecture Description Language* – SMADL.

1. *Vulnerability:* SM networks are naturally less vulnerable to snooping. In an extreme but feasible scenario, each network node manages (the data of) a single user. In this case, a social network of Facebook scale would have a billion plus servers, instead of the approximately two hundred thousand Facebook is believed

to have. In order to collect users' data, all "servers" would have to be accessed, and only those which did not have any constraints around their assets woudl allow that.

2. *Heterogeneity:* SM networks are also more heterogeneous than centralized systems because each of their nodes has the freedom to use any API implementation (protocol) it wishes to. These implementations, in turn, can make use of different service providers such as PaaS, DaaS, cloud storage, etc. to operate. In this context, large scale privacy violations, as supposedly is the case of the PRISM program, would demand "agreements" with a much larger number of service providers than is the case in the current context. Ultimately, pressured by a growing number of privacy violations, users can choose to host their SMs on their own personal computers, making use of their own communication infrastructure.
3. *Use of an Architecture Description Language*: if SMs are created using SMADL, which can handle restrictions of relationships and general interactions, we are able to provide better privacy assurances to all users. SMADL allows operations to be made available under certain constraints, so it leaves us the option of only letting previously known SMs to access those operations. On the other hand, if your application (SM) is requesting service, it must previously define relationships using registered OAuth keys, just as it is necessary to connect you app to Google, Facebook or Dropbox, avoiding ad-hoc connections to unknown or untrusted data providers. In this context, using SMADL reinforces that data privacy protection can be assured.

In the following section we will give an overview of metaContainer, a SM from which we can instantiate heterogeneous and highly distributed SM networks.

4.1 metaContainer, a case study

metaContainer is a SM that is being developed to provide a consumer cloud storage service that ensures its users the preservation, ownership and privacy of their data. It was designed from the vision that people are SMs that implement authentication, PaaS and raw cloud storage APIs and/or hire other SMs such as, respectively, Google, CloudBees and Amazon to provide these implementations from them.

Vision: Creating an account on Google admits two interpretations. From the classic point of view, the user is gaining access to several software products delivered as services. From another point of view and considering people as SMs, Google is being hired to provide implementations of APIs such as email (Gmail), instant messaging (Google Talk), consumer cloud storage (Google Drive), among others, and so allowing people to interact with each other through HTTP and OAuth protocols.

Consumer cloud storage services such as Google Drive and SkyDrive have hundreds of millions of users. This popularity is due, in addition to their intrinsic features, to the fact that these services are provided automatically and free of charge to users as soon as they create an account at Google or Microsoft. An interesting aspect is that, even without knowing it, the user is also hiring an authentication API implementation. The increasingly popular "Sign In With Google" buttons are possible because Google acts as an OpenID protocol identity provider.

PaaS services such as that offered by CloudBees are not used by the general public. The same can be said about raw cloud storage services like Amazon S3. This is expected, as such services were designed to be used by developers. It is interesting to note, however, that the general public, even without being the target of these services, understands the need of them. Most people know, for example, that to use Microsoft Office,

they have to provide an infrastructure (hardware, PC) and platform (Windows). Similarly, most people are familiar with the permissions review and grant process required to install an app in their smartphones.

It is fair to say, therefore, that the view from which metaContainer was designed has still not been fully realized. However, given the tacit knowledge that people have about the need to provide the infrastructure and platform on which an application will be executed, as well as allow it to use these resources, we believe that there are no major obstacles to its full realization. We also believe that scandals such as PRISM will motivate users more sensitive to the preservation, ownership and privacy of their data to take an active role on these issues, such as metaContainer makes possible.

Architecture: metaContainer has two main components, the *Subsidiary* and the *Central*. The first provides a consumer cloud storage from the authentication, DaaS – Database as a Service – and raw cloud storage services of the user. The latter is responsible for deploying the *Subsidiary* component through the authentication and PaaS services of the user. *Subsidiary* and *Central* are completely independent, with no communication between them. The following code snippet shows how SMADL could be used to define MetaContainer.

```
SocialMachine MetaContainer relates to AmazonS3, CloudBees, Google {

  //Every requester application must be authenticated
  general constraint PRE_AUTH_SM

  Relationships {
   googleOpenID with Google [
     uri = "http://www.metacontainer.com/cloudbees"
     api-key = "OIUY2345"
     secret = "ASDEFIUH456H"
     user-token = "2L3KHJ4573456"
   ] type: LIST_OF_OPS (authenticate)

   cloudDBS with CloudBees [
     uri = "http://www.metacontainer.com/google"
     api-key = "IOUQYWERF09871"
     secret = "6546QWEF0712EDIUH"
     user-token = "0981234BVUISDC78G1"
   ] type: LIST_OF_OPS (getApplications, getDatabases, deployApp)

   amazonFileStorage with AmazonS3 [
     uri = "http://www.metacontainer.com/amazon"
     api-key = "98762GRU12QWEG"
     secret = "029384Y5UIHGSDF"
     user-token = "1234123HG65JKHG"
   ] type: LIST_OF_OPS (listFiles, readFileContents, writeFileContents)
  }

  op deployApplication (java.io.File WARFile) {
    /* Gets your app and deploy in our metaContainer reusing web based
       APIs */
  }

  op listCloudFiles () {
    /* List all cloud stored files */
  }
}
```


MetaContainer defined using SMADL.

Central Component: The user, when accessing *Central*, is prompted to enter his OpenID identity. Once the user's identity has been established by his OpenID provider, he may request the deployment of an instance of the *Subsidiary* component by specifying his PaaS, DaaS and raw cloud storage providers and the security information necessary to interact with these providers on behalf of the user. *Central*, in possession of these

informations and the user´s OpenID identity, builds an instance of *Subsidiary* (a Java web application) and deploys it to the platform specified by the user.

Central is also able to migrate a given *Subsidiary* if the user wants to switch his PaaS, DaaS or raw cloud storage providers. In the first case, it is enough to redeploy *Subsidiary* to the new provider. The last two cases are more complex because they require data migration from the old to the new provider.

Subsidiary Component: The *Subsidiary* component provides a consumer storage service from the relational database and raw cloud storage services specified by the user. One aspect that deserves mention in this component is the fact that it is single user, attending only to the user who requested its deployment. This feature manifests itself to the user by the presence of only one "Sign In" button in its welcome page that, when pressed, initiates the authentication process through the OpenID protocol.

Benefits: The metaContainer architecture allows it to be deployed in various configurations. If, for example, the user specifies the metaContainer itself as a provider of PaaS, DaaS and raw cloud storage, metaContainer will be deployed in the SaaS model. Probably this will be the configuration chosen by users who are having contact with metaContainer for the first time and want to assess whether it meets their functional requirements or users who do not have higher expectations about the preservation, ownership and privacy of their data.

At the other extreme, the user can choose to provide, himself, the services required by metaContainer, running on his personal computer, for example, a platform like Micro Cloud Foundry[3]. Most likely this will be the configuration chosen by users with strong preservation, ownership and privacy requirements of their data. These requirements, as well the providers of the services required by metaContainer, may change over time, hence the importance of the Central component be able to migrate the Subsidiary component to a new configuration.

5 Concluding remarks

In this paper, we proposed a new way of designing and implementing networked information systems to favor data ownership and privacy. We outlined some of the problems and consequences of programs like PRISM and provided an introduction to the concept of Social Machines, including its model, examples, a logical interpretation of existing systems and the details of its development language, the Social Machine Architecture Description Language - SMADL. Furthermore, we gave an example of use of the SM model through the development of a case study that integrates the notion of people as Social Machines and software into one composite system called metaContainer. We presented a code snippet with a basic representation of metaContainer using SMADL. We do not tackle all the issues related to privacy and ownership, but, in more than one sense, we think this work contributes to the process of providing feasible ways to make it easier for us to control **OUR OWN** data and, consequently, hide it, if we wish to do so, from prying eyes.

There are many possible future developments. If we are going to cater for ownership and privacy of data and connections, in the web, there ought to be ways, such as the federations of SMs described here, that would allow every single user to take care of his data and establish the properties it wants for it. That would possibly

[3] https://micro.cloudfoundry.com/

mean that the current business model for services such as Facebook would be useless, since there would be no personal data trove at the core of the social network, because the network, well, would be a network, a true graph of nodes –the SMs- and arcs –the relationships.

The net effect of this "novel" architecture for networks would be to send us back to where we were before the internet, from the point of view of gathering information about users, as we have claimed. It would also possibily mean that every user (or every user that would care for his data and actions being private) would have to pay the cost of his networked presence. Not everybody would care to, most probably; would that make those who would care even more targeted by security services around the world? Would Social Machines help to avoid a more Orwellian society or, on the contrary, accelerate the coming of it? We bet on the first, but are prepared to consider arguments for the latter.

Acknowledgments

This work was partially supported by the National Institute of Science and Technology for Software Engineering (www.ines.org.br), funded by CNPq and FACEPE. We would like to thank all colleagues from BACEN (www.bcb.gov.br) and SERPRO (serpro.gov.br) for helping to improve this work.

References

1. Meira, S.R.L.: governo dos EUA vigia todo mundo | dia a dia, bit a bit (in portuguese). TERRA MAGAZINE, available at http://bit.ly/179G582, last accessed in June, 2013. (2013).
2. Meira, S.R.L.: EUA vigia todo mundo: e agora? | dia a dia, bit a bit (in portuguese). TERRA MAGAZINE, available at http://bit.ly/11FY8L3, last accessed in June, 2013. (2013).
3. Solove, D.J.: "I"ve Got Nothing to Hide' and Other Misunderstandings of Privacy. GWU Law School Public Law Research Paper No. 289. Available at SSRN: http://ssrn.com/abstract=998565. 44, 275 (2007).
4. Starr, P.: Waters: Obama Campaign Database Has "Information About Everything on Every Individual" | CNS News. Available at http://bit.ly/121gSDU, last Accessed in June, 2013. (2013).
5. Levinthal, D.: Obama's 2012 campaign is watching you | POLITICO.com. Available at http://politi.co/11f3MaL, Last accessed in June, 2013. (2012).
6. Buregio, V.A.A., Meira, S., Rosa, N.: Social machines: a unified paradigm to describe social web-oriented systems. 22nd International World Wide Web Conference (WWW 2013 Companion). pp. 885–890. International World Wide Web Conferences Steering Committee (2013).
7. Meira, S.R.L., Buregio, V.A.A., Nascimento, L.M., Figueiredo, E., Neto, M., Encarnacao, B., Garcia, V.C.: The Emerging Web of Social Machines. 2011 IEEE 35th Annual Computer Software and Applications Conference. pp. 26–27. IEEE (2011).
8. Buregio, V.A., Meira, S.L., Rosa, N.S., Garcia, V.C.: Moving Towards " Relationship-aware " Applications and Services: A Social Machine-oriented Approach. 17th IEEE International EDOC Conference (EDOCW 2013). p. to be published. , Vancouver, Canada (2013).
9. Roush, W.: Social Machines - Computing means connecting. Technology Review, available at http://bit.ly/1bFHlwy, accessed in June, 2013. 1–18 (2006).
10. Ko, M.N., Cheek, G.P., Shehab, M., Sandhu, R.: Social-Networks Connect Services. Computer. 43, 37–43 (2010).
11. Yuen, M.-C., Chen, L.-J., King, I.: A Survey of Human Computation Systems. 2009 International Conference on Computational Science and Engineering. pp. 723–728. IEEE (2009).

12. Richardson, L., Ruby, S.: RESTful Web Services. O'Reilly Media (2007).
13. Richardson, L., Amundsen, M., Ruby, S.: RESTful Web APIs. O'Reilly Media (2013).
14. Yu, S., Woodard, C.J.: Innovation in the Programmable Web: Characterizing the Mashup Ecosystem. Lecture Notes in Computer Science. 5472, 136–147 (2009).
15. Pressman, R.S.: Software Engineering: A Practitioner's Approach. McGraw-Hill, Inc., New York, NY, USA (2009).
16. Mernik, M., Heering, J., Sloane, A.: When and how to develop domain-specific languages. ACM Computing Surveys (CSUR). 37, 316–344 (2005).
17. Nascimento, L.M., Viana, D.L., Silveira Neto, P.A.M., Souto, S.F., Martins, D.A.O., Garcia, V.C., Meira, S.R.L.: A Systematic Mapping Study on Domain-Specific Languages. Seventh International Conference on Software Engineering Advances (ICSEA 2012). pp. 179–187. , Lisbon, Portugal (2012).
18. Maximilien, E.M., Wilkinson, H., Desai, N., Tai, S.: A Domain-Specific Language for Web APIs and Services Mashups. In: Krämer, B.J., Lin, K.-J., and Narasimhan, P. (eds.) International Conference on Service Oriented Computing–ICSOC 2007. pp. 13–26. Springer (2007).
19. Behrens, H.: MDSD for the iPhone: Developing a Domain-Specific Language and IDE Tooling to produce Real World Applications for Mobile Devices. Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion. pp. 123–128 (2010).
20. Kulkarni, C., Brebner, G., Schelle, G.: Mapping a domain specific language to a platform FPGA. Proceedings of the 41st annual conference on Design Automation DAC 04. pp. 924–927. ACM Press (2004).
21. Medvidovic, N., Taylor, R.N.: A classification and comparison framework for software architecture description languages. IEEE Transactions on Software Engineering. 26, 70–93 (2000).
22. Feiler, P.H., Lewis, B.A., Vestal, S.: The SAE Architecture Analysis & Design Language (AADL) a standard for engineering performance critical systems. Proceedings of the 2006 IEEE Conference on Computer Aided Control Systems Design. pp. 1206–1211. IEEE (2006).
23. Bashroush, R., Spence, I., Kilpatrick, P., Brown, T.J., Gilani, W., Fritzsche, M.: ALI: An Extensible Architecture Description Language for Industrial Applications. 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ecbs 2008). 297–304 (2008).