

Personal APIs as an Enabler for Designing and Implementing People as Social Machines

Vanilson Burégio^{1,3}
¹CIn-UFPE
Informatics Center - Federal
University of Pernambuco
Recife, PE, Brazil
vaab@cin.ufpe.br

Leandro Nascimento^{1,2,3}
Nelson Souto Rosa¹
²DEINFO-UFRPE
Department of Informatics -
Federal Rural University of
Pernambuco
Recife, PE, Brazil
{lmn2,nsr}@cin.ufpe.br

Silvio Meira^{1,2,3}
³INES - National Institute of
Science and Technology for
Software Engineering
Brazil
srlm@cin.ufpe.br

ABSTRACT

In this paper, we extend the initial classification scheme for *Social Machines (SM)* by including *Personal APIs* as a new *SM*-related topic of research inquiry. *Personal APIs* basically refer to the use of *Open Application Programming Interfaces (Open APIs)* to programmatically access information about a person (e.g., personal basic info, health-related statistics, busy data) and/or trigger his/her human capabilities in a standardized way. Here, we provide an overview of some existing *Personal APIs* and show how this approach can be used to enable the design and implementation of people as individual *SMs* on the Web. A proof-of-concept system that demonstrates these ideas is also outlined in this paper.

Categories and Subject Descriptors

H.1.2 [Information Systems]: Models and Principles—
User/Machine Systems
; D.2.0 [Software Engineering]: General—*Standards*

Keywords

Social Machines, Personal APIs, Software Engineering

1. INTRODUCTION

Social Machine (SM) has emerged as a promising multidisciplinary area of research [1], driven by the blending of computational and social processes into Web-enabled systems[2]. In [1], we characterize three convergent visions of the *SM* paradigm: *i) Social Software* - as its foundations; *ii) People as Computational Units* - focuses on research initiatives that integrate people (in the form of human-based computing) and software into one composite system; and *iii) Software as Sociable Entities* - which refers to efforts that address how to weave social elements into software to allow its “socialization”.

As the *SM* paradigm is only beginning to grow, potential scientific inquiries and different approaches are constantly coming up. This fact has lead to the need for reviewing related concepts, technologies and standards that enable us to design and implement social machines. In fact, by exploring the conceptual diagram of the different research visions that are deemed relevant to social machines (as per Burégio et al.[1]), it is possible to extend it through characterizing *Personal APIs* as a proper topic in the *SM* classification scheme, as shown in Figure 1.

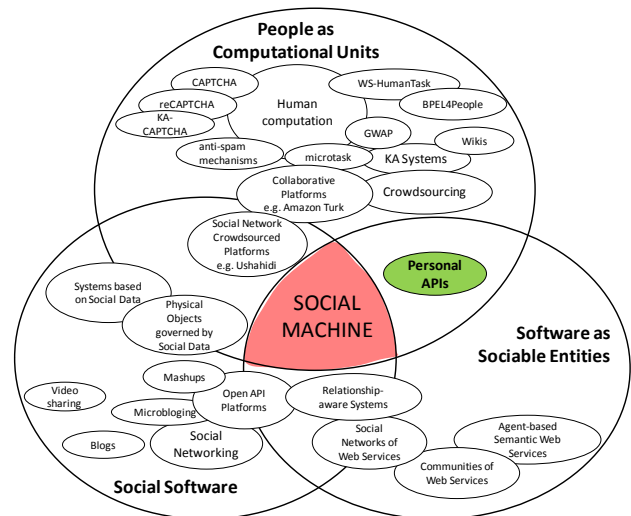


Figure 1: Personal APIs in the converging diagram of the different research visions of social machines

The term *Personal APIs*¹ is used here to designate *Open Application Programming Interfaces (Open APIs)* that allows third-parties to programmatically access information about a person (e.g., *personal basic info*, *health-related statistics*, *busy data*) and/or trigger his/her human capabilities in a standardized way.

We believe that “APIzing” people is also a way of “PERSONifying” software.

¹This term has already been used in the Web industry to specify a set of APIs (<http://api.naveen.com/>) that access real-time personal statistics.

Personal APIs can indeed be used to blend computational and social aspects into Web-enabled systems and, consequently, support the development of certain families of *Social Machines*. Based on this fact, Figure 1 presents **Personal APIs** in the intersection of *People as Computational Units* and *Software as Sociable Entities* visions.

However, despite **Personal APIs** seem to be a potential way of combining computational and social elements into software, a key question emerges: how could we design and implement *SMs* on the Web that really benefit from this approach? In this scenario, this paper has three main contributions: we introduce the notion of **Personal APIs** by presenting some existing initiatives; we incorporate **Personal APIs** in a proper building block used to design people as individual social machines; and we report an implementation experience of creating a personal platform of services as a *SM*.

The remaining sections are organised as follows. Section 2 provides a summary of existing efforts related to **Personal APIs**. Next, Section 3 presents some guidelines to design people as social machines that provide a dynamic set of **Personal APIs**. Section 4 outlines our proof-of-concept and, finally, Section 5 presents some concluding remarks and directions for future work.

2. BACKGROUND

Recently, practical initiatives have wrapped people (their information and/or human capabilities) to provide a set of services through simple, clean, and stable APIs on the Web. We can call these solutions **Personal APIs**. This section provides a brief overview of **Personal APIs** and then discusses some existing practical initiatives.

2.1 Personal APIs

As mentioned before, **Personal APIs** basically refers to the use of *Open Application Programming Interfaces* (**Open APIs**) to allow third-parties to programmatically access information about a person and/or trigger his/her human capabilities. Despite being a fresh topic, in practice, we can find different types of **Personal APIs** in the Web, depending on both the nature of data they deal with and the type of provided services. On one hand, regarding the nature of data, we can highlight different kinds of data such as health-related statistics (e.g., heart rate, weight, sleep patterns), basic personal information (e.g., name, job, age, address), busy data (e.g., agenda, availability, activities) and so on. On the other hand, the provided services range from services that only allow access to wrapped data (through read-only and queryable APIs) to services that expose human capabilities and consequently make possible to request the execution of actions by a person (e.g., request a person to transcript a speech from an audio file into a text document). We briefly present some existing initiatives built upon the concept of **Personal APIs**, namely: The Human API, api.naveen, Personal assistants and VoiceBunny.

2.1.1 The Human API

The **Human API**² use APIs to access personal health data. It allows application developers to create meaning from personal data (e.g., heart rate, active minutes, sleep, genetic makeup or blood glucose) through a simple API. The Hu-

²A platform for human health data, available at <http://humanapi.co>.

man API's data infrastructure collects patient data from different sources and unifies them into a single API by providing analytics-based and dynamic-care experiences for all patients. Each data stream is exposed as an endpoint that can be invoked as a service to build mashups that deal with human health data.

2.1.2 Selvadurai's API

Another example of **Personal API**, namely Selvarudai's API, also tracks health-related data³. This API manages Selvadurai's busy life and periodically sends to his company (or to anyone who desires to be notified) updates about his real-time personal statistics such as weight, sleep time, personal activities, checkins heart rate and others.

2.1.3 Personal Assistants

As a way to improve their communication and engagement with others, some professionals, such as the journalist Brian Proffitt⁴ and the technologist Jay Cousins⁵, have started specifying their own personal interfaces as a set of APIs on the Web. Under some communication constraints, these APIs have been conceived as features of a personal assistant. It seems that, in the future, they expect that such APIs may substitute, for example, a human secretary.

2.1.4 VoiceBunny

As aforementioned, beyond allowing access to personal data, there also have been APIs to trigger human capabilities. In this context, we can highlight **VoiceBunny**⁶ whose **Personal APIs** expose human capabilities as a service on the Web. **VoiceBunny** is a crowdsourced platform that uses thousands of voice actors working from home studios to provide professional "voice as a service" on the Web. It offers RESTful APIs through which is possible to create third-party applications that interact with and consume **VoiceBunny**'s provided services.

As seen, we have set the scenario to discuss **Personal APIs** as a proper research topic, by presenting practical initiatives that come up with potential avenues of applications and development. However, despite seeming a possible way of blending computational and social elements into software, the notion of **Personal APIs** are recent enough to represent difficulties in understanding how they can be efficiently used to support the development of real, practical *Social Machines*. Motivated by this issue, next Section uses the *SM* abstraction model defined in [3] to present how we can design people as *SMs* built upon the concept of **Personal APIs**. Some general guidelines to design *SM*-oriented systems are also presented.

3. DESIGNING PEOPLE AS SOCIAL MACHINES

We have used *Social Machines* as a generic and simple manner to describe and design Web-oriented platforms like Ushahidi, Twitter and Facebook. The unified abstraction

³Naveen Selvadurai's API available at <http://api.naveen.com/>

⁴<http://readwrite.com/2013/08/23/building-personal-api>

⁵<http://jaycousins.wordpress.com/about/personal-api-personality-interface/>

⁶Voice actors over recordings, available at <http://voicebunny.com>

model defined in [3] is the architectural building block native to our design approach for social machines. In this paper, we adopt this abstraction model and incorporate the concept of **Personal APIs** as a way to enable the design of people as “relationship-aware” social machines.

3.1 The Social Machine abstraction model

Our *SM* abstraction 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* [3] that uses the notions from **computing**, **communication** (in the form of *relationships* and *interactions*) and **control**.

As introduced in [3], the abstraction model for 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.”*

Based on this definition, Figure 2a 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 hardware or software. An IPS can be represented, for example, by an algorithm, a Web service, a network of computer processes, or even a person as a “unit of computation”.

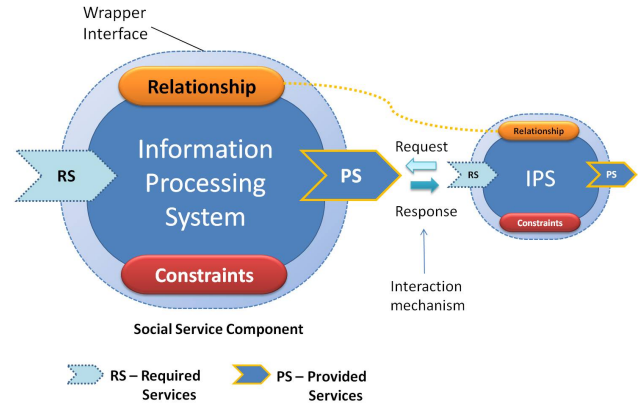
The element *Relationship* is the key concept of this model. It represents a particular type of connection that constrains the way in which two or more *SMs* are associated to or have interactions with each other. It is worth noting that relationships weave “sociability” aspects into software, as a mean of representing and dealing with the proliferation of the emerging “relationship-aware” applications and services, e.g., Facebook.

The *Wrapper Interface*, in its turn, encapsulates the *SM*’s computational unit and provides an interface to be used by the *SM*’s services. It is also responsible for dealing with data (convert, format and so on) that flow from the *SM*’s services to the wrapped computational unit, and vice versa. The *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 them. Optionally, a *SM* can specify *Required Services* (RS).

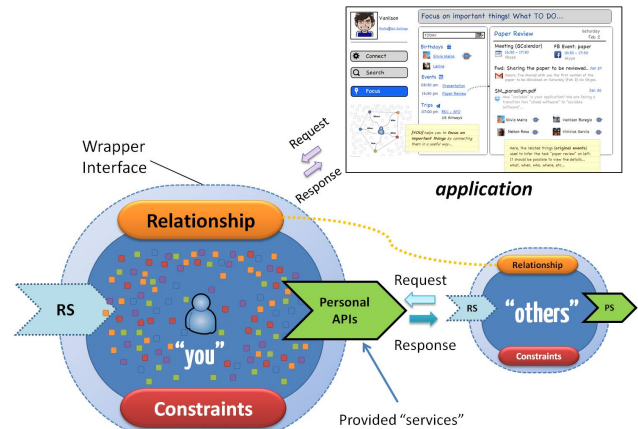
And last, but not least, *Social Machines* have *Constraints* (C), which basically define rules for the establishment of *relationships* and definition of *interaction views* among different *SMs*.

3.2 People as “relationship-aware” SMs

Getting back to the converging diagram of research visions (Figure 1) and adopting our *SM* unified model, it is possible to imagine a movement of the “Personal APIs topic” towards the center or this diagram, i.e. towards the intersection of the three research visions. Assuming this fact, we set up the context to think about possible scenarios involving the use of our *SM* model in conjunction with **Personal APIs**. For



(a) Social Machine as a Social Service Component



(b) People as “relationship-aware” Social Machines

Figure 2: The Social Machine Abstraction Model

example, to consider “you” (the reader, a person) as a kind of computational unit (i.e., an IPS, according to our model). Then, “you” (i.e., your information and/or human capabilities) could be wrapped and represented as an individual social machine in the Web. Figure 2b illustrates this scenario by using the aforementioned *SM* model to represent people as interacting social machines.

As can be seen in Figure 2b, the *SM*’s *Provided Services* (PS) are exposed as **Personal APIs**. Such APIs allow both access to personal data (illustrated in the Figure 2b as little multicolor squares) and the execution of human-based activities.

The set of provided **Personal APIs** is dynamic, i.e., it changes according to who is invoking them. This is a direct effect of the relationship-awareness of *SMs*. In fact, each *SM* represents a person (e.g., “you”) as a *Social Machine* and, like “you”, it provides ways to establish different relationships with “others” (i.e., other people also wrapped as *SMs*). Simply stated, different *relationships* establish different *constraints* and, consequently, different levels of interaction take place between the involved *SMs*.

Constraints (C) of a given *SM* can be specified as a set of rules. For example, one might want to define a rule that establishes different communication priorities for his/her *SM*. Then, in this case, parts of *SM*’s **Personal APIs** will be avail-

able or not, depending on the relationship with who (e.g., family, friend, and so on) is requesting its API.

It is important to highlight that by proving **Personal APIs**, we are defining a software-to-software interface, not a user interface. In fact, the main goal of building autonomous and independently deployed personal *SMs* on the Web is to enable the creation of an ecosystem of applications (see Figure2b) built on top of the services provided by such *SMs*. These applications can enable, for example, large-scale social initiatives using of a multitude of loosely-coupled and distributed personal *SMs*.

We believe that programming personal *SMs* facilitates to launch such kinds of large-scale initiatives on the Web. Undoubtedly, APIs enable the establishment of standard interfaces to communicate with **EVERYone** (in this case) and possibly **anyTHING**, creating the basis for a world in which **EVERYTHING** is going to be socially connected. In more than one sense, this approach can improve the way we build **Social Machines** that indeed combine computational and social aspects into a transparent blending of software, people and perhaps things.

Once we have described our architectural building block, next section presents a general guideline used to support the design of Web-enabled systems based on our *SM* model.

3.3 Design Guidelines

As a software development guideline for designing social machine-oriented architectures, we have considered the following steps: *i) Define building blocks, ii) Specify services, iii) Design data integration and iv) Design interaction models.*

3.3.1 Define building blocks

A key step to design a *SM*-oriented architecture is to define which parts of the system should be socially wrapped. Our high-level abstraction model helps in describing and designing both the whole **Social Machine** as a single *social service component* (e.g., facebook platform as per presented in [3]) as well as parts of the **Social Machine** (e.g., its modules, subsystems and participants) as *relationship-aware entities*. In practice, any entity that establishes *relationships* with others and interacts according to such *relationships* can be wrapped up into a set of specialized APIs, and so described as a relationship-aware entity, following our *SM* abstraction model. These entities include, for example, a source of information, a stateless service (e.g., an URL shorter), a collection of other socially wrapped entities and, as aforementioned, people and their information/behaviors. In a meta-level architecture, our model can also be used to compose existing **Social Machines** into new **Social Machines**. In this way, the obtained system is the result of a set of *SMs* working together. To decide what should be socially wrapped, it is necessary to analyze which parts make sense to involve with a “social layer” in a way that allows the creation of an independent and autonomous entity capable of establishing relationships with others to define its different interaction views. During this design process, each *SM* should have a unique identifier, often a URI, that is used as basis for accessing it’s provided services/APIs.

3.3.2 Specify services

During this step, the set of services provided by the *SM* is defined. The *SM*’s services can be specified in terms of

APIs, including relative URIs, type of request (e.g., GET, POST and so on), possible parameters and description. Often, a common URI syntax is adopted to identify social machines and their services. Table1 shows a simple example of the set of services provided by a personal *SM* called **CalendarYOU**, which is a *SM* that wraps a person’s agenda and provides it as a set of **Personal APIs**.

Table 1: Example of APIs specification

CalendarYOU		
Name	HTTP request	Description
URIs relative to https://{host}/{person}/calendaryou		
<i>link</i>	POST /link	Establishes a relationship with the <i>SM</i> by requesting permission to access the person’s agenda.
<i>details</i>	GET /{id}	Returns the details of a specific agenda identified by {id}.
<i>list</i>	GET /list	Returns the lists of agendas that belongs to the person.
<i>list by date</i>	GET /list?d=yyyyMMdd	Returns the events in a specified date.
<i>search</i>	GET /search?q=query	Returns the events related to the search query.
<i>subscribe</i>	POST /subscribe/{topic}	Subscribes to the topic (i.e., {topic}) of interest.
<i>notify</i>	POST {callback URL}	<i>SM</i> notifies subscribers, via an previously informed callback URL, when an event of interest occurs.

3.3.3 Design data integration

Generally speaking, the process of composing social machines deals with both structured and unstructured data from multiple different sources. As there is a need to integrate heterogeneous data from existing infrastructures, the architecture design of composite social machines encompasses some integration issues. For example, the architecture has to deal with integration issues as collecting and filtering the flows of data from wrapped computational units and/or converting data into common and consistent formats for *SM* manipulation. In this step, a common view of relevant data that occurs when designing *SM*-oriented applications should be defined. Often, a diagram containing the set of abstract data types is used to define the kinds of data manipulated by each *SM*. Once the abstract data types are defined, the mechanisms of converting, mapping and formatting specific data should be designed. It is common to have a combination of different architectural patterns to deal with integrations issues. In our abstraction model, the element *Wrapper Interface* (WI) is often responsible for collecting and converting data from the wrapped computational unit. In order to do that, the implementation of the WI uses the *pipes and filters* architectural style as an integration pattern

to create the logic of data conversion and combination (as per Liu et al. in [4]).

3.3.4 Design interaction models

To define how the components and actions that are part of a system are interrelated is very important to understand and support the real-life user and other application interactions with a system. In the context of social machines, these interactions are intensified due to the large number of possible relationships among the social machine, its users and third-party applications. Because of that, to state the possible interaction models is a fundamental task of the social machine design. An interaction model defines how the interactions among different parts take place. It includes, for example, the definition of communication protocols among these parts and how such parts establish relationships to interact with each other. *Social Machines* can define different interaction models, but some patterns of interactions can be observed and are worth highlighting. One of these patterns is called by some the “third-party authentication” model, which is accomplished by an authentication model involving the *SM*, its users and third-party applications.

The third-party authentication model is a generalization of the *oauth* model [5] and defines three roles: *API Provider* (the *Social Machine*), *Data Owner* (user) and *API Consumer* (third-party applications), as illustrated in Figure 3.

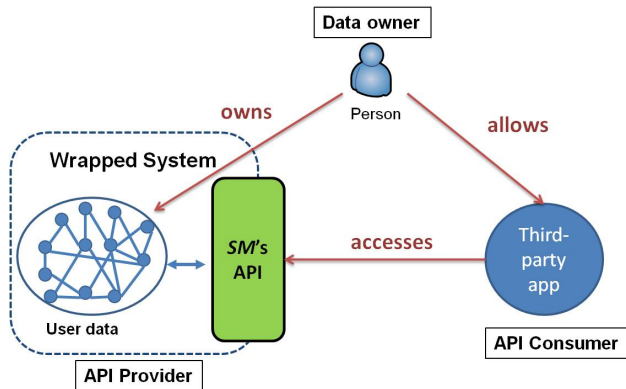


Figure 3: The third-party authentication model

As it can be seen in Figure 3, the *SM* wraps a system, which keeps the user data, and exposes a set of APIs to be accessed by third-party applications. Such applications should be previously registered in the social machine and some credentials should also be established in advance (see *link* in Table 1). In this model, third-party applications act on behalf of a user. They do not access their own data but those of the user (the data owner). In this case, an application should use the data owner’s credentials to make requests. The third-party authentication interaction model is commonly adopted when a *SM* exposes users’ data in the form of a platform of APIs to be accessed by other applications, which is normally the case of personal *SM*s. In this scenario, there is a need to establish an authentication model that involves these different parts. Currently, this model is supported by protocols for authentication and authorization like *OpenID* and *OAuth* [5], which recently have gained wide popularity in the Web.

4. IMPLEMENTING “YOU” AS A SM

In this section, we describe the implementation of a system, namely [*YOU*], as a proof-of-concept that uses the initial ideas of people as *Social Machines*. This proof-of-concept represents a relevant starting point to apply the aforementioned design guidelines and creates an example that embraces the notion of *Personal APIs*.

The central motivation of [*YOU*] is the fact that nowadays we have to deal with a large number of information about (and related to) us. In general, this information is spread across multiple sources in the Web (e.g., Facebook, Dropbox, Twitter, Google Services, and so on) and, consequently, we have to do a huge effort to connect related things that matter to us. Then, one of the goals behind the [*YOU*] *SM* is to provide a single access point to your information with the possibility of connecting and sharing them in a useful way.

4.1 Implementation Outline

The [*YOU*] *SM* was implemented as a *personal information retrieval platform* in which “you” (the information related to you) is wrapped as a composite *Social Machine*. On top of the [*YOU*]’s services, we built the [*YOU*] application - a Web-based interface for the [*YOU*] *SM* (as per the model illustrated in Figure 2b). The adopted approach allows to consume the [*YOU*]’s services with third-party applications that makes viable the creation of an ecosystem of applications built on top of its services. Figure 4 shows an overview of the [*YOU*] *SM*’s architecture.

As shown in Figure 4, we designed the whole system as a composite social machine internally formed by the combination of the user’s different sources of information (i.e., Web platforms like Facebook, GCalendar, Dropbox and others). Each source of information was wrapped as an independent and autonomous *SM*. In this way, it was possible to independently deploy each one on a different provider.

The services provided by [*YOU*] and its internal *SM*s were designed as endpoints of a REST API. A set of common services was defined for each social machine, like the ones already presented in Table 1.

The [*YOU*] *Social Machine* as a whole uses a combination of different architectural styles (i.e., *data federation*, *pipes and filters*, and *MVC*) to aggregate and combine data and functions from the existing sources. The relation between these styles is illustrated in Figure 4.

In general, the [*YOU*] ecosystem follows the *Model-View-Control* (MVC) style as it manipulates data to present views according to user inputs. In this case, the model has access to data from the set of individual *Social Machines*, and groups them in a structure to be used by the [*YOU*]’s provided services.

The *Data Federation* style was used as a way of aggregating and correlating the necessary data from multiple sources. As can be seen in Figure 4, the data federation is realized by the set of parallel *pipes and filters* defined by each individual *SM*. In this style, the source data remains under control of the individual social machines and are asynchronously pulled on demand for federated access.

This process is made by the *Information Grouping* component that acts as an asynchronous data handler, enabling the [*YOU*] *SM* to start an “external process” (such as a searching in the *CalendarYOU SM*) while the handler continues processing. Then, the handler can continue without waiting for the external process to finish. This design deci-

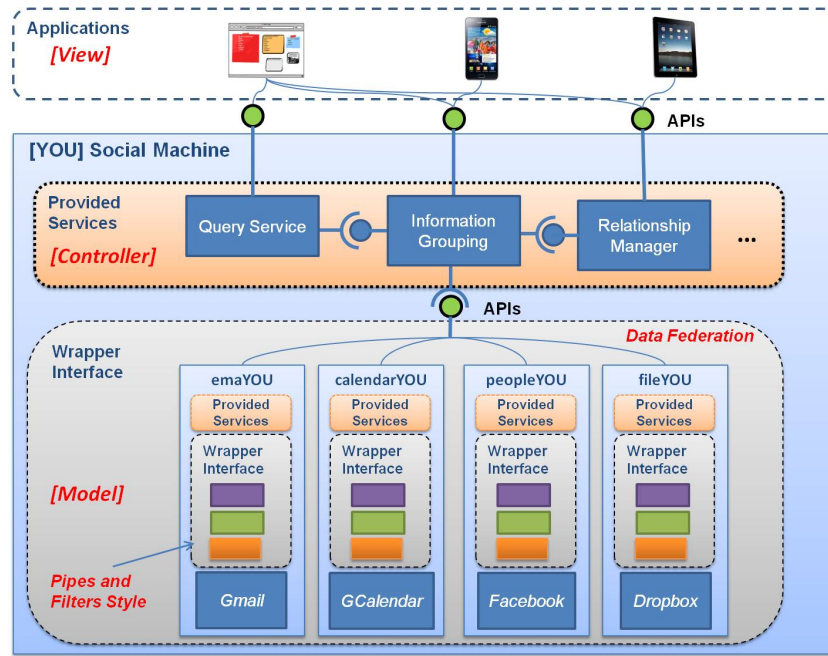


Figure 4: [YOU] SM: A personal information retrieval platform

sion was very significant to solve performance issues faced during the process of data federation.

The [YOU] SM really embraces the idea of **Personal APIs**. With its implementation we could realize several factors that should be taken into account when designing a SM as a platform of services. For example, integration issues were substantially minimized by combining the adopted design patterns. Using single services (e.g., link) to abstract the flows of authorization and authentication greatly reduced the complexity of implementing such processes. Also, the role of the *wrapper interface* became much more clearer with the introduction of converting and formatting operations.

5. CONCLUSIONS

In this paper, we identified **Personal APIs** as a new SM-related topic of research. Existing solutions are discussed and some guidelines to design people as social machines (making use of **Personal APIs**) are also presented. To illustrate the practical usage of the proposed approach, a proof-of-concept of developing a personal SM is presented. In practice, one can say that “implementing” people, their countless kinds of real relationships and interactions sounds like orders of magnitude above the complexity of implementing common software. Indeed, we are in the infancy of this implementation. However, in more than one sense, we believe that this work contributes to accelerate the process of transforming the relationships among people, software and possibly things. “Implementing” people as SMs enables the creation of a fully connected environment in which the concepts of virtual/concrete, people/software totally merge. In this environment, the possibilities of interactions among nodes (i.e., SMs) to create large-scale social initiatives go far beyond and above the sort of interactions we have today. We think it is time to start caring about people by themselves as individual SMs, as a way to naturally support crowdsourced platforms on the Web and, consequently,

enable the establishment of a society in which the notions of computing and “being social” will be indeed completely blended.

6. ACKNOWLEDGMENTS

This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES⁷), funded by CNPq and FACEPE, grants 573964/2008-4 and APQ-1037- 1.03/08.

7. REFERENCES

- [1] Vanilson Buregio, Silvio Meira, and Nelson Rosa. Social machines: A unified paradigm to describe social web-oriented systems. In *Proceedings of the 22Nd International Conference on World Wide Web Companion*, WWW '13 Companion, pages 885–890, 2013.
- [2] Nigel Shadbolt. Knowledge acquisition and the rise of social machines. *International Journal of Human-Computer Studies*, 71(2):200–205, February 2013.
- [3] Vanilson Buregio, Silvio Meira, Nelson Rosa, and Vinicius Garcia. Moving towards relationship-aware applications and services: A social machine-oriented approach. In *17th IEEE International EDOC Conference Workshops (EDOCW)*, 2013, pages 43–52, 2013.
- [4] Yan Liu, Liang Xin, Lingzhi Xu, Mark Staples, and Liming Zhu. Using architecture integration patterns to compose enterprise mashups. In *2009 Joint Working IEEE/IFIP Conference on Software Architecture & ECSA*, pages 111–120. IEEE, September 2009.
- [5] Barry Leiba. OAuth Web Authorization Protocol. *IEEE Internet Computing*, 16(1):74–77, January 2012.

⁷www.ines.org.br